Sparse Polynomial Interpolation and Testing

by

Andrew Arnold

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philsophy
in
Computer Science

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Interpolation is the process of learning an unknown polynomial f from some set of its evaluations. We consider the interpolation of a *sparse* polynomial, i.e., where f is comprised of a small, bounded number of terms. *Sparse interpolation* dates back to work in the late 18th century by the French mathematician Gaspard de Prony, and was revitalized in the 1980s due to advancements by Ben-Or and Tiwari, Blahut, and Zippel, amongst others. Sparse interpolation has applications to learning theory, signal processing, error-correcting codes, and symbolic computation. Closely related to sparse interpolation are two decision problems. *Sparse polynomial identity testing* is the problem of testing whether a sparse polynomial f is zero from its evaluations. *Sparsity testing* is the problem of testing whether f is in fact sparse.

We present effective probabilistic algebraic algorithms for the interpolation and testing of sparse polynomials. These algorithms assume *black-box* evaluation access, whereby the algorithm may specify the evaluation points. We measure algorithmic costs with respect to the number and types of *queries* to a black-box oracle.

Building on previous work by Garg–Schost and Giesbrecht–Roche, we present two methods for the interpolation of a sparse polynomial modelled by a *straight-line program (SLP)*: a sequence of arithmetic instructions. We present probabilistic algorithms for the sparse interpolation of an SLP, with cost softly-linear in the *sparsity* of the interpolant: its number of nonzero terms. As an application of these techniques, we give a multiplication algorithm for sparse polynomials, with cost that is sensitive to the size of the output.

Multivariate interpolation reduces to univariate interpolation by way of *Kronecker substitution*, which maps an n-variate polynomial f to a univariate image with degree exponential in n. We present an alternative method of *randomized Kronecker substitutions*, whereby one can more efficiently reconstruct a sparse interpolant f from multiple univariate images of considerably reduced degree.

In error-correcting interpolation, we suppose that some bounded number of evaluations may be erroneous. We present an algorithm for error-correcting interpolation of polynomials that are sparse under the Chebyshev basis. In addition we give a method which reduces sparse Chebyshev-basis interpolation to monomial-basis interpolation.

Lastly, we study the class of Boolean functions that admit a sparse Fourier representation. We give an analysis of Levin's Sparse Fourier Transform algorithm for such functions. Moreover, we give a new algorithm for testing whether a Boolean function is Fourier-sparse. This method reduces sparsity testing to homomorphism testing, which in turn may be solved by the Blum–Luby–Rubinfeld linearity test.

Acknowledgements

I would like to acknowledge some of the people who supported me throughout my studies:

- My supervisor, Mark Giesbrecht. Mark has been very generous with his time and resources; introduced inspiring problems to me; and gave me the freedom and support to pursue my research. Mark has provided me direction and opened many doors for me along my career path. Thank you, Mark.
- My friend Dan Roche, for our many collaborations. It is always a pleasure discussing mathematics with Dan. I hope we continue to work together on interesting problems.
- Eric Blais and Erich Kaltofen, for the opportunity to work with each of them on rewarding projects that became chapters of my thesis.
- My committee, for their thoughtful questions, and their careful reading of my thesis draft. In addition to other committee members mentioned herein I would like to name my internal-external examiner Kevin Hare; my external examiner Jeremy Johnson; and Éric Schost, whose work on interpolation was a starting point for my doctoral research. Thanks as well to Evelyne Hubert and Clément Pernet for sharing stimulating questions and ideas.
- My Master's supervisor, Michael Monagan, without whose direction I would not have pursued doctoral studies at Waterloo; and Peter Borwein, who first encouraged me to pursue research.
- To my friends at the Symbolic Computation Group (SCG) lab, who made it a fun and stimulating place to work at: Curtis Bright, Reinhold Burger, Shaoshi Chen, Mustafa Elshiek, Joseph Haraldson, Albert Heinle, Ilias Kosteras, Winnie Lam, Roman Lebreton, Steve Melczer, Vijay Menon, Andy Novocin, Colton Pauderis, and Nam Phan. I also thank SCG faculty George Labahn and Arne Storjohann for the times they've given me valued professional advice.
- For their generous financial support: the National Sciences and Engineering Research Council of Canada (NSERC), the Ontario Graduate Scholarship (OGS) program, the David R. Cheriton School of Computer Science, the University of Waterloo, and the Fields Institute.
- My friends, who enriched my time spent in Waterloo and Toronto: Parsiad Azimzadeh, Cecylia Bocovich, Duane Bobbsemple, Hicham El-Zein, Sam Gentile, Hella Hoffmann, Mark Horton, Carol Jung, Stephen Kiazyk, Erik Postma, Shadab Rashid, Johl Ringuette, Dean Shaft, Valerie Sugarman, Jack Thomas, Oliver Trujillo, The Clockwork (my ultimate team), and the David R. Cheriton School of Hockey Superfriends (my intramural hockey team); also my out-of-town friends Kyle Allesia, Jamieson Anderson, Colin Beaumont, Nate Hapke, Reuben Heredia, Athena Li, Chelsea Richards, Mike Rozen, and Rob Szolomicki, for being a phone call away whenever I needed a laugh.

Lastly, I want to thank my family: my parents, Dave and Marie; my brother Thomas; and my sister-in-law Yoshiko. I could not have written this thesis without their encouragement and support.

To my parents and my big brother Thomas.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis explores efficient methods of *polynomial interpolation*: the process of learning a polynomial f from its evaluations. By learning f, we mean learning some useful representation of f, e.g., a linear combination of powers of variables. Astronomers used polynomial interpolation to predict the position of celestial bodies, as early as the time of ancient Babylon; and was developed further by the likes of Newton, Waring, and Lagrange during the scientific revolution [Mei02]. Interpolation is often thought of as a numerical problem; however, we intend to study the problem from an algebraic perspective. Interpolation is a cornerstone of symbolic computation, and has applications to computational complexity, error-correcting codes, scientific computing, and signal processing. One may also obtain evaluations from signal measurements, in which case the aim of interpolation is to find a polynomial that nicely approximates the signal.

The focus of this thesis is on sparse polynomials. We usually say a polynomial or a signal is *sparse* if it admits a natural concise (i.e., sparse) representation, or is well-approximated by such a representation. Otherwise we say that a polynomial is *dense*. For example, the polynomial $2x^0 + 3x^{99}$ could be naively represented by a dense representation $(2, 0, 0, \ldots, 0, 3)$ of length 100, or one could represent it with the sparse representation $((2, 0), (3, 99))$. We say a polynomial or signal is T**-sparse** if it can be represented as a linear combination of T basis elements. In many cases the bit size of the sparse representation may be as small as logarithmic compared to the size of the dense representation.

Sparse signals occur naturally in a variety of contexts. Lossy compression schemes such as JPEG and MPEG rely on sparsity occurring in video, image, and audio signals. Biomedical signals admit sparsity that allows for wireless medical monitoring devices to consume fewer resources [Sma]. Sparse signals also occur in computational learning theory [KM93], data centre monitoring [MNL10], and database similarity search [AFS93].

Sparse interpolation algorithms exploit sparsity in order to recover the interpolant polynomial more efficiently than a naive dense algorithm. Sparse interpolation falls under the larger

umbrella of algorithms that exploit sparsity in a variety of contexts. Sparse signals and sparse systems have become a widely-studied topic over the past decade, with recent advancements such as compressed sensing [Don06], sparse Fourier transforms [Has+12], and randomized numerical linear algebra [Sar06]. In many settings we can solve problems in polynomial-time, and sometimes softly-linear-time, in the size of the underlying sparse representation. One challenge of achieving such a runtime is that it imposes strong bounds on the number of polynomial evaluations or signal measurements one can make. In many cases sparse algorithms only inspect a fraction of their inputs; such algorithms, including many herein, make random choices.

One of our goals is to study and improve upper bounds on the asymptotic complexity of sparse polynomial arithmetic. Using fast interpolation/evaluation schemes such as the Fast Fourier Transform, arithmetic of dense polynomials can be performed in time that is softly linear in the combined sizes of the inputs and outputs. We give a number of algorithms that allow for the fast computation of *straight-line programs (SLPs)* that output sparse polynomials, for univariate and multivariate cases and for functions over either an arbitrary commutative ring with identity, or a finite field. We also give an asymptotically fast algorithm for the multiplication of sparse polynomials with integer coefficients.

We mention here two decision problems closely related to sparse interpolation. *Sparsity testing* is the problem of deciding whether a polynomial f is T-sparse from its evaluations. Sparsity testing is a useful subroutine to estimate the sparsity of f, which itself is a useful and often necessary preprocessing step to run before a sparse interpolation algorithm. Sparsity testing allows us to estimate the sparsity of f, a necessary preprocessing step to sparse interpolation. *Sparse polynomial identity testing (sparse PIT)* is the problem of determining whether $f = 0$, given that f is T-sparse. We collate a number of sparsity-testing and sparse-PIT algorithms throughout literature, as well as give an algebraic algorithm for Fourier-sparsity testing on *Boolean* functions over the hypercube.

1.1 Preliminaries

This thesis assumes the reader has a literacy in algebra, algorithms and asymptotic notation, and probability theory. Table 1.1 gives an unexhaustive list of some of the objects we will commonly use throughout this thesis. We expect that the reader is familiar with these objects, some of which we define in the introduction.

We list some basic conventions of this thesis. We will write vectors in lower-case boldface,

and scalars in normal-weight font, including vector entries. E.g., a vector $\boldsymbol{s} \in \mathbb{Z}^n$ may be written

$$\boldsymbol{s} = (s_1, \ldots, s_n) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} = \left[s_i\right]_{i=1}^{t},$$

whereas $\boldsymbol{s}_i \in \mathbb{Z}^n$ would be a vector indexed by i. We write $\boldsymbol{s}_i = (s_{i1}, \ldots, s_{in})$. Indices will be only be comma-separated when it usefully aids readability, e.g. $a_{i+1,j-1}$. We will typically reserve capitalized bold-faced font for matrices, e.g., we may write

$$\mathbf{R} = \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \vdots \\ \boldsymbol{r}_m \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \ldots & r_{1m} \\ r_{21} & r_{22} & \ldots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \ldots & r_{nm} \end{bmatrix}$$

In the case that $m = n$ we may write $\mathbf{R} = [r_{ij}]_{i,j=1}^{n}$. We let $\mathbf{R}^\top$ denote the transpose of $\mathbf{R}$.

Given $\boldsymbol{r}, \boldsymbol{s} \in \mathsf{R}^k$ and $a \in \mathsf{R}$, we let $a\boldsymbol{s} = (as_1, \ldots, as_k)$ and $\boldsymbol{r}\boldsymbol{s} = (r_1 s_1, \ldots, r_k s_k)$. Given $\boldsymbol{\omega} \in \mathsf{R}^k$ for a ring R and $\boldsymbol{e} \in \mathbb{N}^k$, we let $\boldsymbol{\omega}^{\boldsymbol{e}}$ mean either $(\omega_1^{e_1}, \ldots, \omega_n^{e_n})$ or $\prod_{i=1}^{n} \omega_i^{e_i}$, depending on the context. We also use shorthand notation for multivariate expressions, e.g., $f(\boldsymbol{\omega}) = f(\omega_1, \ldots, \omega_n)$. These conventions are described in more detail in Section 1.3 and Chapter 5. We will generally use caligraphic fonts to denote sets, e.g., $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$. Here and throughout, we let $0 \in \mathbb{N}$[†].

1.2 Model of Computation

We will often describe our algorithms in terms of *soft-oh* notation.

Definition 1.2.1 (soft-oh notation). *Given two functions $f, g : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$, we write $f \in \widetilde{\mathcal{O}}(g)$ if and only if $f \in \mathcal{O}(g \log^c g)$ for some constant $c \in \mathbb{Z}$. For instance, $\log n \log\log^2 n \in \widetilde{\mathcal{O}}(\log n)$. If $f(n) \in \widetilde{\mathcal{O}}(n)$ we say f is* **softly linear** *in n. We also define $\mathrm{polylog}(g)$ and $\mathrm{poly}(g)$ to denote functions such that $\mathrm{polylog}(g) \in \mathcal{O}(\log^c g)$ and $\mathrm{poly}(g) \in \mathcal{O}(g^d)$ for some constants $c, d \in \mathbb{Z}$.*

We also define $\widetilde{\Theta}, \widetilde{\Omega}$, and $\widetilde{o}$ as "soft" analogues to asymptotic classes of functions given by Θ, Ω, and o, respectively. These are defined in the obvious manner.

We use *soft-oh* notation in order to make our analysis less cumbersome, while examining the most significant factors appearing in a cost function. We note that $f \in \widetilde{\mathcal{O}}(g)$ implies that $f \in \mathcal{O}(g^{1+\epsilon})$ for any constant $\epsilon > 0$.

[†]As it should be!

Table 1.1: Some conventions and definitions of this thesis

$x, y, z, x_1, \ldots, x_n$	variables, i.e., indeterminates
f, g, h	polynomials
$\boldsymbol{f}$	a vector polynomial (Sec. 1.3.1)
τ	a term
$\deg(f)$; $\deg_x(f)$	total degree of f; degree of f with respect to $\boldsymbol{x}$
$\|f\|_1$; $\|f\|_\infty$	the *length* and *height* of $f \in \mathbb{Z}[x]$
C	upper bound on the height of $f \in \mathbb{Z}[x]$
$\#f$	sparsity of f, i.e., number of terms in f
$D; D^*$	bounds on the partial and total degrees of f
$T; n$	bounds on sparsity of f, and the number of variables of f
$\mathcal{S}$	a finite set
$\#\mathcal{S}$	number of elements of $\mathcal{S}$
$\mathsf{R}; \mathsf{Z}; \mathsf{K}$	commutative ring with identity; integral domain; field
$\mathsf{R}[\boldsymbol{x}] = \mathsf{R}[x_1, \ldots, x_n]$	ring of polynomials over R with indeterminates $x_1, \ldots, x_n$
$f(\boldsymbol{x})$	a multivariate polynomial $f(x_1, \ldots, x_n) \in \mathsf{R}[\boldsymbol{x}]$
$\mathsf{R}[\boldsymbol{x}^{\pm 1}] = \mathsf{R}[x_1^{\pm 1}, \ldots, x_n^{\pm 1}]$	ring of *Laurent polynomials* over R
$\mathsf{R}[\boldsymbol{x}]_D$	R-module of polynomials with partial degrees at most D
$\langle a_1, \ldots, a_m \rangle$	ideal generated by $a_1, \ldots, a_m \in \mathsf{R}$
$\langle \boldsymbol{g} \rangle$	$\langle g_1, \ldots, g_m \rangle$, where $\boldsymbol{g} = (g_1, \ldots, g_m) \in \mathsf{R}[x]^m$
$f^{\bmod p}$	$f(\boldsymbol{x}) \bmod \langle x_1^p - 1, \ldots, x_n^p - 1 \rangle$, for $f \in \mathsf{R}[\boldsymbol{x}]$
$\mathsf{R}[\boldsymbol{x}]^{\bmod p}$	The quotient ring $\mathsf{R}[\boldsymbol{x}]/\langle x_1^p - 1, \ldots, x_n^p - 1 \rangle$
μ	a probability bound, typically a parameter for algorithms
$\mathbb{Z}; \mathbb{Z}_{>0}; \mathbb{N}$	ring of integers; positive integers; natural numbers
$\mathbb{Z}_m$	ring of integers modulo $m \in \mathbb{Z}_{>0}$
$\mathbb{Q}; \mathbb{R}; \mathbb{C}$	fields of rational, real, and complex numbers
$\mathbb{F}_q$	A finite field of size q, q not necessarily prime
$[m]$	the integer range $\{1, 2, \ldots, m\}$, where $m \in \mathbb{Z}_{>0}$
$[k..m]$	For $k < m \in \mathbb{Z}$, the range of integers $\{k, k+1, \ldots, m\}$
$(k..m); (k..m]; [k..m)$	$[k..m]$ with k and m, m, and k removed, respectively
$\mathbb{V}(v_1, \ldots, v_n)$	$n \times n$ *Vandermonde matrix* parameterized by values $v_1, \ldots, v_n$
$\mathbb{H}(h_1, \ldots, h_{2n-1})$	$n \times n$ *Hankel matrix* parameterized by values $h_1, \ldots, h_{2t-1}$
$\mathbb{T}(t_{-n+1}, \ldots, t_{n-1})$	$n \times n$ *Toeplitz matrix* parameterized by values $t_{-n+1}, \ldots, t_{n-1}$

Many of our algorithms are **Monte Carlo**, meaning that their runtime is deterministic but they fail with nonzero probability at most μ, for some parameter $\mu > 0$. Monte Carlo algorithms make random choices, and require a source of randomness, e.g., a string of random bits, which it will use to decide what choices to make. Such algorithms usually admit a soft-oh $\log(1/\mu)$ factor in its cost.

We note that an algorithm that produces a unique output upon success, and succeeds with probabilty at least $1 - c$ for some constant $c > \frac{1}{2}$, can be made to succeed with probability $1 - \mu$ for arbitrarily small $\mu > 0$ by running the original algorithm $\mathcal{O}(\log(1/\mu))$ times and producing the

most frequently occuring output of the original algorithm. This is described in detail in Section 2.3.

We will also work with **Las Vegas** algorithms, which are randomized algorithms that always produce a correct result but whose runtime is probabilistic. Any Las Vegas algorithm can be made Monte Carlo by terminating the algorithm, regardless of whether it has completed, after an appropriate deterministic length of time. By Markov's inequality, a Las Vegas algorithm with an expected runtime at most $f(n)$ on inputs of size n will terminate in time $kf(n)$ time with probability at least $1 - 1/k$.

We will usually assume a **probabilistic algebraic random access machine (PARAM)** as our model of computation (see Section 2 of [Kal88] for an exact definition), though we do not intend to encumber our analysis with too many model-specific details. This is a modification of the random access machine (RAM) described in [AHU74]. Like the RAM, the probabilistic algebraic RAM has an infinite array to store words, allow for pointer manipulation, and to maintain a stack; however, this array also allows for the storage and manipulation of elements of a specified ring R. The PARAM can copy, compare, add, substract, and multiply ring elements, and division by units (i.e., invertible elements). The PARAM in addition has a source of randomness, such that an algorithm running on a PARAM can choose $x \in \mathcal{S}$ uniformly at random for any specified finite set $\mathcal{S}$.

In some cases when we are specifically interested in the bit complexity on a PARAM, where we consider the bit cost arithmetic operations over a specified ring R (we will call these simply R-**operations**). A PARAM usually assumes either *unit-cost* or *logarithmic-cost* memory access, whereby the cost of accessing the nth element of either array is $\mathcal{O}(1)$ or $\mathcal{O}(\log n)$ respectively. The choice of whether to use a unit-cost or logarithmic-cost RAM will not affect a soft-oh algorithmic runtime analysis, provided the runtime is polynomial in the memory cost, or greater.

We may measure our algorithms in terms of bit operations, R-operations, polynomial evaluations, or some combination thereof. In the event that an algorithm has a cost of $\mathcal{O}(f(n))$ R operations, we will assume an implied cost of $\mathcal{O}(f(n))$ bit operations as well.

In some cases, we will consider algorithms for which the output size may vary asymptotically for inputs of a fixed size. In such cases we may measure the complexity of the algorithm in terms of the **problem size**, by which we mean the combined bit-size of the inputs and outputs.

This thesis does not explore parallel algorithms. We remark though that parallel sparse interpolation of multivariate *black-box polynomials* or *extended black-box polynomials* (Section 1.4) have been studied, e.g., in [GKS90] and [HR99]. Parallel sparse black-box polynomial interpolation over finite fields has been studied (and implemented) in [JM10].

1.3 Sparse and dense polynomial representations

The word polynomial comes the Greek words *poly* (many) and *nomós* (portion). In a computer algebra system, the most commonly used polynomial representations express a polynomial f as a sum of its parts, i.e. *monomials* or *terms*. The **dense representation** expresses a univariate $f \in \mathsf{R}[x]$ as a list of terms of consecutive degree

$$f(x) = \sum_{i=0}^{D} c_i x^i, \quad D \geq \deg(f), \tag{1.1}$$

or in the n-variate case we may write

$$f(x_1, \ldots, x_n) = \sum_{\boldsymbol{e} \in [0..D]^n} c_{\boldsymbol{e}} x_1^{e_1} \cdots x_n^{e_n} \in \mathsf{R}[x_1, \ldots, x_n], \tag{1.2}$$

which we may condense to

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{e} \in [0..D]^n} c_{\boldsymbol{e}} \boldsymbol{x}^{\boldsymbol{e}} \in \mathsf{R}[\boldsymbol{x}].$$

We will typically reserve n for the number of variables in our polynomial ring $\mathsf{R}[\boldsymbol{x}]$.

In some instances we may consider partial degree bounds $D_1, \ldots, D_n$, whereby $D_i \geq \deg_{x_i}(f)$, and we write f as a linear combination $\boldsymbol{x}^{\boldsymbol{e}}$ over all $\boldsymbol{e} \in \mathbb{Z}_{\geq 0}^n$ with $e_i \leq D_i$ for $i = 1, 2, \ldots, n$. In other cases we may consider a bound D^* on the **total degree** of f. The total degree of a nonzero term $c\boldsymbol{x}^{\boldsymbol{e}}$ is $\sum_{i=1}^n e_i$. The total degree of f is the maximum of the total degree of its terms.

In computer memory, one can represent (1.1) as a vector $(c_0, \ldots, c_D)$. One may similarly represent (1.2) as a multi-dimensional array indexed by $[0..D]^n$. The advantage of such a representation is that degree of each monomial $c_i x^i$ is implicitly given by the array index. An advantage of the dense representation over the monomial basis is that it lends itself to asymptotically fast and, in practise, efficient arithmetic and interpolation in a wide variety of settings.

However, if most of the coefficients of f are zero, it may be more natural to use a **sparse representation**,

$$f(x) = \sum_{i=1}^{t} c_i x^{e_i}, \quad \text{or} \quad f(\boldsymbol{x}) = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} \tag{1.3}$$

in the univariate and multivariate cases respectively, where the *exponents* e_i (or $\boldsymbol{e}_i$) are distinct and sorted and the c_i are explicitly nonzero. In the univariate case we suppose $e_1 > e_2 > \cdots > e_t$. In the multivariate case, we will suppose the $\boldsymbol{e}_i$ are sorted with respect to the lexicographical ordering $\succ$, where for $\boldsymbol{d}, \boldsymbol{d}' \in \mathbb{Z}_{\geq 0}^n$, $\boldsymbol{d} \succ \boldsymbol{d}'$ if and only if there exists $k \in [n]$ such that $d_\ell = d'_\ell$ for all $\ell \in [1..k)$, and $d_k > d'_k$. We call the exponent components e_{ij} of $f \in \mathsf{R}[\boldsymbol{x}]$ the *partial exponents*

of f. We will assume throughout this thesis that the $\theta(\log D)$ bits are used to store every partial exponent of a sparse polynomial f. Thus the sparse representation of f takes $\mathcal{O}(T \log D)$ bits and $\mathcal{O}(T)$ ring elements.

We note that the size of a dense representation (1.2) of $f(x)$ is $(D+1)^n$ ring elements, whereas the sparse representation (1.3) requires $\mathcal{O}(T)$ ring elements and another $\mathcal{O}(Tn \log D)$ bits in order to store the exponents of f. As the number of terms T is at most $\mathcal{O}(D^n)$, the bit size of a dense representation is soft-oh asympotically at least that of the sparse representation.

We will generally let a term of f denote a nonzero term and a **formal term** denote a term $c\boldsymbol{x}^e$ of f, where c may possibly be zero, and reserve **term** to mean a nonzero term. We will often write τ to denote a term. We also will use the short-hand notation $\tau \in f$ to mean that τ is a term of f.

In the PARAM model, we would store the expression (1.3) as a list $[(c_i, e_i)]_{i=1}^t$, and similarly in the multivariate case with the exponents sorted with respect to the lexicographical ordering. We identify f with the set of its terms, and write $\tau \in f$ to mean that τ is a term of f.

We say that f a is **sparse polynomial** if it has significant "gaps" between terms of consecutive degree, otherwise we will say f is a **dense polynomial**. We also let a sparse (dense) polynomial refer to a polynomial given by its sparse (dense, resp.) representation, though it is generally assumed that a polynomial f given by its sparse representation admits such gaps, such that it is well suited towards a sparse representation. The **sparsity** of f is its number of nonzero terms. We denote by $\#f$ the sparsity of f, and say f is T**-sparse** if $\#f \leq T$.

Sparse polynomials are also called **lacunary polynomials** and **fewnomials** in literature. As with dense representations, sparse representations naturally generalize to multivariate polynomials and alternative bases. We will also refer to a polynomial as sparse (dense) if it is given by its sparse (dense, respectively) representation.

Given a sparse representation of $f \in \mathsf{R}[x]$, in order to construct its dense representation we initialize a dense array with each entry $0 \in \mathsf{R}$, and then populate the array with the nonzero coefficients of f while making a pass of the sparse representation. We can similarly construct a sparse representation of f from its dense representation by making a single pass through the dense representation, and appending a term to the sparse representation of f whenever we come across a nonzero coefficient. We conclude the following lemma.

Lemma 1.3.1. *We can convert between the sparse and dense representations of f with bit cost linear in the combined size of the representations.*

For $f = \sum_{i=1}^t c_i x^{e_i} \in \mathbb{Z}[x]$, we define the **height** of f to be $\|f\|_\infty \stackrel{\text{def}}{=} \max_{i=1}^t |c_i|$ and the **length** of f to be $\|f\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^t |c_i|$.

We will also consider **Laurent polynomials**, which allow for negative integer exponent entries as well. That is, we allow for exponents of the form

$$f = \sum_{\boldsymbol{e} \in [-D..D]^n} c_{\boldsymbol{e}} \boldsymbol{x}^{\boldsymbol{e}} \in \mathsf{R}[\boldsymbol{x}^{\pm 1}] \stackrel{\text{def}}{=} \mathsf{R}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}] \tag{1.4}$$

We will define, for $f \in \mathsf{R}[x^{\pm 1}]$, the **absolute degree** of f, $\operatorname{absdeg}(f) \stackrel{\text{def}}{=} \max_{e \in \operatorname{supp}(f)} |e|$, as well as partial absolute degrees defined in a similar fashion.

A sparse or dense representation for F is with respect to a particular *basis* for $\mathsf{R}[\boldsymbol{x}]$ as an R-module. We most often use the **monomial basis**, comprised of the set of all monomials $\{\boldsymbol{x}^{\boldsymbol{e}} : \boldsymbol{e} \in \mathbb{Z}_{\geq 0}^n\}$. We will also consider the univariate **Chebyshev basis**, comprised of the **Chebyshev polynomials** T_i, $i \in \mathbb{Z}_{\geq 0}$, defined by

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{i+2}(x) = 2xT_{i+1}(x) - T_i(x), \quad i \geq 0.$$

The multivariate Chebyshev basis is similarly comprised of products of univariate Chebyshev polynomials in each variable, e.g., $T_{e_1}(x_1)T_{e_2}(x_2)\cdots T_{e_n}(x_n)$.

The sparse interpolation problem, roughly, will be to construct a sparse representation of f given some model of evaluation for f. We discuss some of these models in Section 1.4.

1.3.1 Vector polynomials

In some cases we will work with **vector polynomials**. We will say $\boldsymbol{f}$ is a length-m vector polynomial if it may be written as

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})) \in \mathsf{R}[\boldsymbol{x}]^m,$$

where $f_i \in \mathsf{R}[\boldsymbol{x}]$ for $i = 1, \ldots, m$. We identify $\mathsf{R}[\boldsymbol{x}]^m$ with $\mathsf{R}^m[\boldsymbol{x}]$ in a natural manner, writing $\boldsymbol{f}$ as a sum of the form

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_e \boldsymbol{c}_{\boldsymbol{e}} \boldsymbol{x}^{\boldsymbol{e}},$$

where $\boldsymbol{c}_{\boldsymbol{e}} \in \mathsf{R}^m$, and $\boldsymbol{c}\boldsymbol{x}^e$ is understood to be taken as $(c_1\boldsymbol{x}^{e_1}, \ldots, c_n\boldsymbol{x}^{e_n})$. One can consider sparse and dense representations of vector polynomials.

An n-variate vector polynomial induces a map $\Phi : \mathsf{R}^n \to \mathsf{R}^m$ given by

$$(x_1, \ldots, x_n) \mapsto \boldsymbol{f}(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)).$$

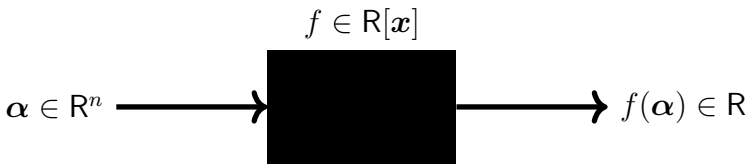

Figure 1.1: A black-box for a polynomial f

1.4 Models of polynomial evaluation

In fast dense interpolation methods the set of evaluation points used is typically fixed. For instance the **fast Fourier transform (FFT)** allows for the fast evaluation and interpolation of $f \in \mathsf{R}[x]$, using specific **roots of unity** as evaluation points, i.e. elements ω from R or an extension thereof, for which $\omega^n = 1$ for some $n \in \mathbb{Z}_{>0}$. The **order** of ω is the least positive integer n such that $\omega^n = 1$. The FFT allows for fast evaluation and interpolation with roots of unity as evaluation points, allowing for practical and fast dense polynomial arithmetic.

However, sparse interpolation algorithms often require that evaluation points are chosen randomly. We consider three models of evaluations, in order of increasing power: black-box polynomials, extended-black-box polynomials, and straight-line programs. A straight-line program can simulate an extended black-box polynomial, which can simulate a black-box polynomial, which in turn can produce a set of evaluations over a fixed set of points.

1.4.1 Black-box polynomials

We will often think of such an algorithm as being given a "black box" which encapsulates the interpolant f, allowing an algorithm to evaluate f without letting us see its monomial representation.

Definition 1.4.1. *A* **black box** *for a polynomial $f \in \mathsf{R}[x_1, \ldots, x_n]$ is an oracle that, when queried with $\boldsymbol{\alpha} \in \mathsf{R}^n$, produces $f(\boldsymbol{\alpha})$. We call a polynomial given by a black box a* **black-box polynomial**.

Figure 1.1 gives a graphical representation of a black-box polynomial. Prony's algorithm (Sec. 2.7) interpolates sparse, black-box polynomials over integral domains containing roots of unity with order exceeding the degree of f. Black-box interpolation algorithms are measured in terms of the number of black-box **queries**, i.e., evaluations produced by the black-box, in addition to bit and ring operations.

In some cases we may restrict the set of α with which we may query our black box. For instance, in Chapter 7 we will study functions acting over the n-dimensional **hypercube**, $\mathbb{F}_2^n$.

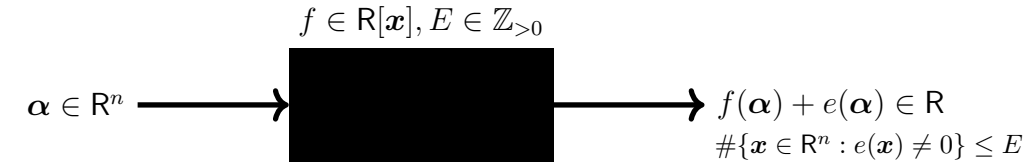

Figure 1.2: An erroneous black-box for a polynomial f

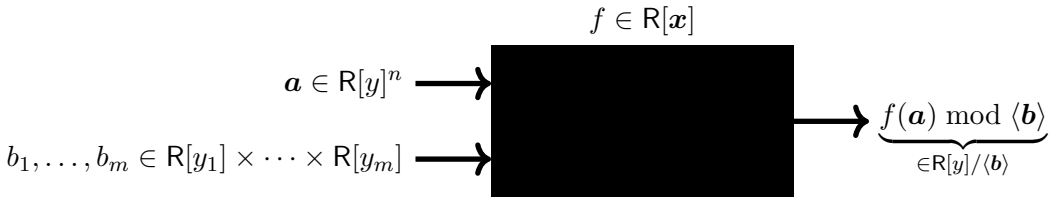

Figure 1.3: An extended black-box for a polynomial f

We will identify the additive group structure of $\mathbb{F}_2$ with the multiplicative group structure of $\{\pm 1\} \subset \mathbb{R}$, and consider a black-box polynomial $f \in \mathbb{R}[x_1, \ldots, x_n]$ that may only be evaluated over $\{\pm 1\}^n$. Restricting to these evaluation points, f admits a reduced representation modulo $(x_i + 1)(x_i - 1), i \in [n]$. That is, f admits a multilinear represenation. We consider the problem of testing whether f is sparse with respect to this approximation.

1.4.1.1 Erroneous-black-box polynomials

In Chapter 6 we will consider an **erroneous black box** that may produce either an evaluation of f or an erroneous evaluation. In this model, we suppose that the black-box may produce an incorrect evaluation for f at up to E evaluation points, for some bound E. We further suppose that the black-box will produce a fixed output for any evaluation point, such that if the black box produces an incorrect value for some evaluation $f(\boldsymbol{\alpha})$, repeated queries will not reveal the true value of $f(\boldsymbol{\alpha})$. We can think of the black box as producing $f(\boldsymbol{\alpha}) + e(\boldsymbol{\alpha})$, where e is an error function that is zero except on a subset of R^n of cardinality at most E.

This model was introduced by Comer, Kaltofen, and Pernet in [CKP12], where the authors studied interpolation with erroneous evaluations.

1.4.2 Extended-black-box polynomials

In some cases, black-box access to a polynomial $f \in \mathsf{R}[x]$ will be insufficient to reconstruct a monomial representation of f. For instance, if we are given black-box access to $f \in \mathbb{F}_q[x]$, we can

only interpolate f modulo $\prod_{\alpha\in\mathbb{F}_q}(x-\alpha) = x^q - x$. Thus, for instance, the polynomials x^2 and x^{q+2}, evaluated at any $\alpha \in \mathbb{F}_q$, will agree, even though they are distinct polynomials.

We extend the notion of a univariate black-box to allow for evaluation in certain extensions of R that allow us to interpolate f over R with arbitrarily many terms.

Definition 1.4.2. *Let* R *be a commutative ring with identity, and* Z *an integral domain.*

- *An* **extended black box** *for* $f \in \mathsf{R}[x_1, \ldots, x_n]$ *is an oracle that, queried with* $a_1, \ldots, a_n \in \mathsf{R}[y_1, \ldots, y_m]$ *and a monic polynomials* $b_i \in \mathsf{R}[y_i]$*, for* $i \in [m]$*, produces*

$$f(\boldsymbol{a}) \bmod \langle \boldsymbol{b} \rangle \stackrel{\text{def}}{=} f(a_1, \ldots, a_n) \bmod \langle b_1, \ldots, b_n \rangle, \tag{1.5}$$

  where $\mathcal{B} = \langle b_1, \ldots, b_m \rangle \subseteq \mathsf{R}$.

- *An* **algebraic black box** *for* $f \in \mathsf{Z}[x_1, \ldots, x_n]$ *is an extended black-box with the additional restriction that the* b_i *are* **separable***, meaning that* b_i *has no multiple roots in the algebraic completion of* Z.

We say f *given by an extended black-box an* **extended black-box polynomial***, and define* **algebraic black-box polynomials** *similarly.*

Extended black boxes allow for the interpolation of univariate polynomials of arbitrary degree and number of nonzero terms. An algebraic black box is a natural generalization of a black box. An algebraic black box may be implemented provided evaluation access to $f \in \mathsf{Z}[\boldsymbol{x}]$ over the algebraic closure of the field of fractions of Z. One can reconstruct $f(\boldsymbol{a}) \bmod \langle \boldsymbol{b} \rangle$ by evaluating f at $x_i = a_i(\beta_{ij})$, where β_{ij} is taken over the roots of b_i, for $i \in [m]$. One could conceivably further generalize an algebraic black box to allow us to take images of f modulo a radical zero-dimensional ideal $\mathcal{I} \subset \mathsf{Z}[y_1, \ldots, y_n]$. For the purposes of our algorithms we do not require this generality.

In the case that f is over an algebraically closed field K, a black box for f is an algebraic black box. Thus interpolation algorithms designed for an extended black box may work, for instance, on a black-box polynomial over $\mathbb{C}$, though in that setting numerical considerations exist.

In the case that $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, an extended black-box can evaluate f over field extensions of $\mathbb{F}_q$. Namely, if we fix an irreducible polynomial $\Phi \in \mathbb{F}_q[y]$ of degree d, we can evaluate f at $\boldsymbol{\omega} \in \mathbb{F}_{q^d} \cong \mathbb{F}_q[y]/\langle \Phi \rangle$. We can also construct images of f over $\mathbb{F}_{q^d}$, e.g., for some $g(x) \in \mathbb{F}_q[x]$,

$$f(\boldsymbol{\alpha} x) \bmod (\Phi, g(x)) \in \mathbb{F}_q[x, y]/\langle \Phi, g(x) \rangle \cong \mathbb{F}_q[x]/\langle g \rangle.$$

We will measure the cost of an interpolation algorithm of an extended black box on the number of queries to the extended black-box polynomial f, as well as in terms of the types of inputs $\boldsymbol{a}, \boldsymbol{b}$ used to query f. Here we may need an appropriate cost function for producing various extended-black-box queries. Chapter 3 discusses this in greater detail.

1.4.2.1 Randomized black-boxes

A **randomized black-box** polynomial $f \in \mathsf{R}[\boldsymbol{x}]$ is an oracle whereby the algorithm does not query f with an evaluation point $\boldsymbol{x}$. Rather, the algorithm queries f and gets back a pair $(\boldsymbol{x}, f(\boldsymbol{x}))$, where $\boldsymbol{x}$ is chosen according to a distribution (e.g., uniform) over a domain $\mathcal{D} \subset \mathsf{R}$.

We do not consider this model in the thesis but remark that the corresponding sparse interpolation problem has been studied, for instance, in the case that $f \in \mathbb{R}[\boldsymbol{x}]$, where queries are taken over the **real hypercube** $\{\pm 1\}^n$ in [And+14] and [Koc+14].

1.4.3 Straight-line programs

Definition 1.4.3. *A* **straight-line program (SLP)** *acting on variables $x_1, \ldots, x_n$ over a ring R is a list of arithmetic instructions $\boldsymbol{\Gamma} = (\Gamma_1, \ldots, \Gamma_L)$, where each instruction Γ_i is of the form*

$$\Gamma_i : \beta_i \leftarrow \gamma_{i1} \star \gamma_{i2},$$

where $\star \in \{+, -, \times, \div\}$, and

$$\gamma_{i1}, \gamma_{i2} \in \mathsf{R} \cup \{x_1, \ldots, x_n\} \cup \{\beta_1, \ldots, \beta_{i-1}\}, \quad \textit{for} \quad i \in [L].$$

We let L denote the **length** *of $\boldsymbol{\Gamma}$ and say $\boldsymbol{\Gamma}$ is n-***variate***. We say β_L is the* **output** *of $\boldsymbol{\Gamma}$.*

In other words, every straight-line program instruction assigns to the value β_i the result of an binary arithmetic operation acting on some combination of ring constants, variables and previously computed and stored algebraic values. A straight-line program may be represented by a *directed acyclic graph (DAG)* with labelled edges, whereby every vertex represents either a variable, ring constant, or arithmetic instruction. A vertex is connected by a directed edge to an instruction vertex if it is an operand to that instruction. The source vertices are precisely the variables $x_1, \ldots, x_n$ and ring constants that are instruction inputs. Each node representing an arithmetic operation instruction has in-degree 2. For nodes representing a subtraction or division operation, we label its in-edges 1 or 2 to specify whether an input is the first or second operand. We may reuse a value arbitrarily many times, such that the out-degree of a node is only limited by the number of subsequent arithmetic-operation nodes. Figure 1.4 illustrates an example straight-line program. The computer algebra software DAGWOOD of Freeman et al. was specifically designed for computation with straight-line programs [Fre+88].

We will assume that every variable x appearing in a straight-line program appears in an instruction (or else we could trivially disregard x), such that $n \in \mathcal{O}(L)$.

A straight-line program is also referred to in literature as an **arithmetic circuit** or **white-box polynomials**. We say $\boldsymbol{\Gamma}$ *computes* $f \in \mathsf{R}(x_1, \ldots, x_n)$ if the value assigned to β_i by the instruction

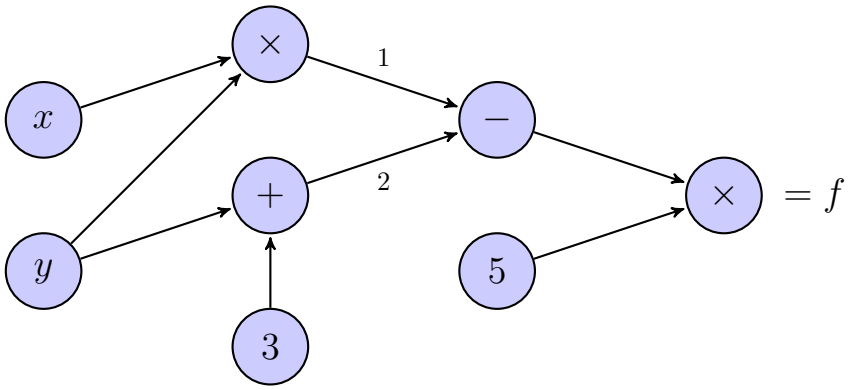

Figure 1.4: A straight-line program for $f = 5(xy - (y + 3))$

Γ_i is well-defined for all i, and γ_L produces the output f. We will sometimes write $\mathbf{\Gamma}^f$ to denote an SLP that computes f.

The straight-line program is a very powerful model, allowing for evaluation over arbitrary ring extensions of R. For any ring extension $\mathsf{R}_1 \supset \mathsf{R}$ and any $\boldsymbol{\alpha} \in \mathsf{R}_1^n$, we evaluate $\mathbf{\Gamma}$ at $(x_1, \ldots, x_n) = (\alpha_1, \ldots, \alpha_n)$ by replacing each instance of x_i appearing in $\mathbf{\Gamma}$ with α_i, before performing the resulting sequence of arithmetic operations. The output may be undefined for a given evaluation point if an arithmetic instruction calls for division by a zero divisor. We will restrict our attention to division-free SLPs, such that the output is always defined.

More generally, we may think of an SLP as a sort of "homomorphic black box", allowing us to construct arbitrary homomorphic images of f. Given any ring homomorphism Φ mapping the polynomial ring $\mathsf{R}[x_1, \ldots, x_n]$ to another ring R_2, we may construct $\Phi(f)$ by applying Φ to the value of any source vertex in the DAG representation of $\mathbf{\Gamma}^f$, i.e., each variable and constant appearing in $\mathbf{\Gamma}^f$. We let $\Phi(\mathbf{\Gamma}^f)$ denote this transformation of $\mathbf{\Gamma}^f$. The resulting arithmetic instructions of $\Phi(\mathbf{\Gamma}^f)$ are over R_2. The output of $\Phi(\mathbf{\Gamma}^f)$ is $\Phi(f)$. For instance, if we take R_2 to be $\mathsf{R}[y_1, \ldots, y_m]/\langle b_1, \ldots, b_m \rangle$ for some $b_i \in \mathsf{R}[y_i], i \in [m]$, and let Φ be an R-linear map given by $x_j \mapsto a_j$ for some $a_j \in \mathsf{R}[x_1, \ldots, x_n], j \in [n]$, then the output of $\Phi(\mathbf{\Gamma}^f)$ is precisely $f(\boldsymbol{a}) \bmod \langle \boldsymbol{b} \rangle$. Thus a straight-line program may simulate an extended black-box polynomial.

We may even transform a straight-line program $\mathbf{\Gamma}$ by maps other than ring homomorphims, such as inverse homomorphims. E.g., one could take an SLP acting over $\mathsf{R} = \mathbb{Z}_m$, and, for $j \in [0..m)$, identify $j \bmod m \in \mathbb{Z}_m$ with j. As with black-boxes we will let a **query** to an SLP computing f refer to the construction of an image of f with respect to a map Φ. As with extended black boxes, we will need to consider the number and types of queries made by an algorithm in a cost analysis of that algorithm.

1.5 Organization of thesis

In Chapter 2, we provide some of the tools we use within sparse interpolation. These include asymptotically fast algorithms from dense polynomial arithmetic, integer and finite field arithmetic, linear algebra, and number theoretic operations such as finding and testing primes. We also cite some basic probabilistic inequalities and methods used in probabilistic algorithms. We also discuss the DeMillo–Lipton–Schwartz–Zippel Lemma and the problem of dense PIT of a black-box polynomial. We also give the theory behind Prony's algorithm for evaluating a black-box polynomial, as well as the method of Kronecker substitution, which reduces multivariate interpolation to univariate interpolation.

In Chapter 3, we give two Monte Carlo algorithms for the sparse interpolation of a univariate polynomial given by an extended black box or a straight-line program. We give two algorithms, one for polynomials over arbitrary rings and an improved algorithm in the case that the interpolant f is over an integral domain with sufficiently many elements. This builds on work by Garg and Schost, and Giesbrecht and Roche. We apply some of the techniques developed for these algorithms in Chapter 4, where we give an asympotically fast algorithm for the multiplication of sparse polynomials. This algorithm is probabilistic and sensitive to the size of the output, which may vary widely for inputs with fixed parameters. As a subroutine to the multiplication algorithm, we give a fast probabilistic algorithm for constructing the *sumset* of two finite sets of vectors of integers $\mathcal{A}, \mathcal{B} \subset \mathbb{Z}^n$, i.e., the set of all sums $a + b$ where $a \in \mathcal{A}$ and $b \in \mathcal{B}$.

In Chapter 5, we give algorithms for the interpolation of sparse multivariate polynomials. These algorithms reduce multivariate interpolation to univariate interpolation, by reconstructing the interpolant f from a number of randomly-selected univariate images. We call this technique *randomized Kronecker substitutions*, and it may work with any black-box algorithm over a sufficiently large integral domain. "Classical" Kronecker substitution, in comparison, reconstructs multivariate f deterministically from a single univariate image. In the case that f is sufficiently sparse, randomized Kronecker substitution can outperform multivariate interpolation via classical Kronecker substitution. We give an additional algorithm that combines ideas from Chapter 3 with randomized Kronecker substitutions, giving an algorithm for the sparse interpolation of an extended black-box or SLP with an improved runtime when f is highly multivariate.

In Chapter 6, we consider the interpolation of a univariate polynomial f that is sparse under the Chebyshev basis, where f is given by an erroneous black-box. We give two algorithms, one that generalizes a black-box sparse interpolation algorithm by Lakshman and Saunders, and one that reduces interpolation in the Chebyshev basis to interpolation in the monomial basis. In Chapter 7, we consider the problem of sparsity testing. Specifically, we consider the specific case of testing whether a **boolean function** f, a function acting over the hypercube, admits a sparse representation as a real-valued multilinear polynomial.

This thesis is based on work from the following six publications:

- (with Mark Giesbrehct and Daniel S. Roche) Faster sparse interpolation of straight-line programs. Proceedings of the 14th International Workshop on Computer Algebra in Scientific Computing (CASC 2013). [AGR13]
- (with Mark Giesbrecht and Daniel S. Roche) Sparse polynomial interpolation via low-order roots of unity. Proceedings of the 40th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014). [AGR14]
- (with Daniel S. Roche) Multivariate sparse interpolation using randomized Kronecker substitutions. ISSAC 2014. [AR14]
- (with Erich Kaltofen) Error-correcting sparse interpolation for the Chebyshev basis. ISSAC 2015. [AK15]
- (with Daniel S. Roche) Output-sensitive algorithms for sumset and sparse polynomial multiplication. ISSAC 2015. [AR15]
- (with Mark Giesbrecht and Daniel S. Roche) Faster sparse multivariate polynomial interpolation of straight-line programs. Journal of Symbolic Computation. [AGR15]

Chapter 3 is based on work from [AGR13] and [AGR14]; Chapter 4 from [AR15]; Chapter 5 from [AR14] and [AGR15]; and Chapter 6 from [AK15]. Chapter 7 is part of currently unpublished joint work with Eric Blais.

Given the dependencies between chapters, we recommend the thesis be read in the following manner:

- Chapters 1 and 2 first;
- Chapter 3 before Chapters 4 and 5;

though Chapter 2 is mostly a tabulation of results, and can be skipped and referred to as needed when reading later chapters.

If I have seen further, it is by standing on the shoulders of giants.

Issac Newton

Chapter 2

Algorithmic tools

In this chapter we will collate some of the basic tools and methods we will use as subroutines in subsequent algorithms. These tools include:

- fast arithmetic over the integers, modular rings, finite fields, and dense polynomials, all of which rely heavily on the Fast Fourier Transform (FFT);
- fast multipoint evaluation and its inverse operation, simultaneous modular reduction;
- fast linear algebraic subroutines;
- efficient data structures for dictionaries;
- basic probabilistic inequalities and methods for probabilistic algorithms;
- number theoretic subroutines for constructing primes or random sets of primes of from a specified range.

Lastly, we give some well-known tools from sparse interpolation: Kronecker substitution, reducing multivariate interpolation to univariate interpolation; Prony's algorithm for interpolating a black-box polynomial; and the technique of early termination, which we use for *sparsity estimation*.

2.1 Integer, dense, modular ring, and finite field arithmetic

To that end we collate a few fundamental results. Most of these appear in *Modern Computer Algebra* by Joachim von zur Gathen and Jürgen Gerhard [VG03], the "bible" of computer algebra.[1] We will, however, give references to the source material as well wherever possible.

2.1.1 Integer and modular arithmetic

Theorem 2.1.1. *Let $m, n \in \mathbb{Z}$. The following can be computed in time softly-linear in the bit-length of the inputs:*

- $m \pm n, mn, m \operatorname{rem} n$*, for any* $m, n \in \mathbb{Z}$*;*
- *Arithmetic in* $\mathbb{Z}_m$ *for any* $m \in \mathbb{Z}_{>0}$.

Again, addition and subtraction trivially achieves this cost via the grade-school method. Division with remainder reduces to multiplication via Newton iteration (Thm. 9.8, [VG03]). All subquadratic multiplication methods essentially rely on interpolation-evaluation methods.

Fast integer multiplication has an interesting history. In the 1950's, Andrey Kolmogorov conjectured that multiplication of two n-bit integers requires $\mathcal{O}(n^2)$ bit operations. In 1960, Kolmogorov stated this conjecture in a Moscow University seminar, attended by Anatolii Alexeevitch Karatsuba. Karatsuba subsequently derived a $\mathcal{O}(n^{\log_2 3})$ bit-operation algorithm based on the divide-and-conquer paradigm. In 1962, Komolgorov wrote [KO63], containing Karatsuba's algorithm and a separate result by Yuri Ofman. Karatsuba only later became aware of this publication (see his personal account in [Kar95]). Subsequently, Andrei Toom and Stephen Cook generalized Karatsuba's method to give a family of multiplication algorithms that divide the problem into $2k-1$ equally sized multiplications, for arbitrary, fixed $k > 1$, with runtime $\mathcal{O}(n^{\log_{2k-1} k})$, essentially achieved $\mathcal{O}(n^{1+\epsilon})$ for arbitrarily large ϵ. Karatsuba's algorithm is merely the case $k = 2$. A considerable caveat of Toom–Cook multiplication is that the constant hidden within "$\mathcal{O}$" grows very fast with respect to k (see Section 9.5.3, [CP06]).

Multiplication of two n-bit integers can be done in $\mathcal{O}(n \log n \log\log n)$ bit operations due to the method of Schönhage and Strassen [SS71]. Most high-performance implementations of fast integer arithmetic use this method (e.g., GMP [GG14]). The Schönhage–Strassen method maintained the fastest asymptotic runtime until in 2009 when Fürer gave an algorithm with a runtime $\mathcal{O}(n \log n 2^{\mathcal{O}(\log^* n)})$ [Fr09] on a multitape Turing machine, where $\log^* n$ is the *iterated logarithm*, given by $\log^* x = 0$ for $x \leq 1$ and $\log^* = 1 + \log^*(\log x)$ otherwise. De et al. gave a

[1] Many also appear in the "Old Testament", *Algorithms for Computer Algebra* by Geddes, Czapor, and Labahn [GCL92]

version of Fürer's algorithm based on modular arithmetic in [De+13]. More recently, Harvey, van der Hoeven, and Lecerf gave a refined analysis of Fürer's algorithm, and showed that it achieved a bit complexity of $\mathcal{O}(n \log n 16^{\log^* n})$. In addition, they gave a slightly improved algorithm with runtime $\mathcal{O}(n \log n 8^{\log^* n})$. In the unit-cost RAM model, Schönhage–Strassen multiplication and its contemporaries all admit a runtime of $\mathcal{O}(n \log n)$.

2.1.2 Dense polynomial arithmetic

Theorem 2.1.2. *Let* R *be a ring, and* $f, g \in \mathsf{R}[x]$ *be given by their dense representations, where* $\deg(f), \deg(g) \leq D$*. Then the following can all be computed with* $\widetilde{\mathcal{O}}(D)$ *ring operations and similarly many bit operations:*

- $f \pm g, fg$*;*
- $f \bmod g$*, if* R *is commutative and with identity, and* g *is monic;*
- $f \bmod g$*, if* R *is a field.*
- $\gcd(f, g)$*, if* R *is a field.*

Addition and subtraction achieve such a runtime using the grade-school method. In order to achieve this cost for multiplication, one can use the algorithm of Cantor and Kaltofen [CK91], which performs a base-2 or base-3 FFT over an appropriate extension of R, for a cost of $\mathcal{O}(n \log n)$ multiplications and $\mathcal{O}(n \log n \log\log n)$ additions and substractions. In the case that $\mathsf{R} = \mathbb{F}_p$, a finite field of prime size p, Harvey, van der Hoeven, and Lecerf have improved the bit complexity using faster FFT-based techniques in the spirit of work by Fürer and De et al. Fast division, as in the integer case, reduces to fast multiplication via Newton iteration (Thm. 9.6, [VG03]).

Computing $\gcd(f, g)$, the greatest common divisor (GCD) of $f, g \in \mathsf{K}[x]$, via the Euclidean algorithm may take $\mathcal{O}(D^2)$ operations. The Euclidean algorithm produces a **remainder sequence** $(r_0, r_1, \ldots, r_k) \in \mathsf{K}[x]^{k+1}$ defined by

$$r_0 = f, \quad r_1 = g, \quad r_{i+2} = r_i \bmod r_{i+1} \quad \text{for } i \in [0..k-2], \quad r_{k-1} \neq r_k = 0. \tag{2.1}$$

is the remainder of r_i divided by r_{i+1}, for $i \geq 0$. If $r_k = 0$, then $\gcd(f, g) = r_{k-1}$.

The GCD may be expedited to $\widetilde{\mathcal{O}}(D)$ arithmetic operations over arbitrary Euclidean domains using the **half-GCD (HGCD) algorithm**. Given $f, g \in \mathsf{K}[x]$, with $\deg(f) > \deg(g)$, the HGCD algorithm produces a unimodular matrix $\mathbf{Q} = [q_{ij}]_{i,j=1}^2 \in \mathsf{K}[x]^{2\times 2}$ satisfying

$$\begin{bmatrix} r_\ell \\ r_{\ell+1} \end{bmatrix} = \mathbf{Q} \begin{bmatrix} f \\ g \end{bmatrix}, \qquad \text{where } \deg(r_\ell) \geq \deg(f)/2 > \deg(r_{\ell+1}),$$

and whose components are individually of minimal degree.

The HGCD algorithm was developed by Schönhage for integers [Sch71], improving on a fast integer GCD algorithm of Knuth. Moenck generalized the HGCD algorithm to polynomial rings [Moe73]. Moenck only gave an analysis of his algorithm on inputs that produced "normal" remainder sequences: sequences such that $\deg(r_{i+1}) = \deg(r_i) - 1$ for all i. An analysis of Moenck's algorithm on arbitrary sequences was given in [AHU74]; however, Thull and Yap noted this analysis was incorrect, and provided a corrected version of the HGCD algorithm in with a proof of correctness in [TY]. The HGCD algorithm is a valuable subroutine in the study of linearly generated sequences (see Section 2.7).

Theorem 2.1.3. *There exists an algorithm that, given dense polynomials $f, g \in \mathsf{K}[x]$, produces r_ℓ and $r_{\ell+1}$ appearing in remainder sequence defined by (2.1).*

2.1.3 Finite field operations

Throughout this thesis, we will assume a finite field written as $\mathbb{F}_q$, where $q = p^r$ for some prime p, is represented as the quotient ring $\mathbb{Z}_p[y]/\langle\Phi\rangle$, where $\Phi \in \mathbb{Z}_p[y]$ is an irreducible polynomial of degree r.

It follows from Theorems 2.1.1 and 2.1.2 that we achieve the following cost for finite field arithmetic.

Theorem 2.1.4. *Let p be prime and $q = p^r$. Then we can perform arithmetic operations in $\mathbb{F}_q$, represented as $\mathbb{Z}_p[y]/\langle\Phi\rangle$ in $\widetilde{\mathcal{O}}(r \log p) = \widetilde{\mathcal{O}}(q)$ bit operations.*

In some cases we will be given evaluation access to a polynomial over $\mathbb{F}_q$, and we will need to evaluate f over an extension $\mathbb{F}_{q^s}$. We will similarly represent $\mathbb{F}_{q^s}$ as $\mathbb{F}_q[w]/\langle\Psi\rangle$, where $\Psi \in \mathbb{F}_q[w]$ is of degree s and irreducible over $\mathbb{F}_q$. We cite the following results, which allows us to quickly construct irreducible polynomials.

Theorem 2.1.5 ([Sho94], [CL13])**.** *There exist Las Vegas algorithms that, given an irreducible polynomial $\Phi \in \mathbb{Z}_p[y]$ of degree r, produces an irreducible polynomial $\Psi \in (\mathbb{Z}_r[y]/\langle\Phi\rangle)[w]$ of degree s with costs:*

- *$\widetilde{\mathcal{O}}(s^2 + s \log q)$ bit operations;*
- *$\mathcal{O}(s^{1+\epsilon(s)}(\log q)^{5+\epsilon(q)})$ bit operations, where $\epsilon : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{>0}$ satisfies $\lim_{n\to\infty} \epsilon(n) = 0$.*

2.1.4 Chinese remaindering and simultaneous modular reduction

Often we will reconstruct exponents from their images modulo a choice of primes. One can apply Chinese Remaindering iteratively to reconstruct an exponent less than D in $\widetilde{\mathcal{O}}(\log^2 D)$ steps. We achieve a better complexity via the following result.

Theorem 2.1.6 ([BM74]). *Given $m_1, \ldots, m_r \in \mathbb{Z}_{>1}$, $m = \prod_{i=1}^{r} m_i$ and $v_i \in \mathbb{Z} \cap [0, m_i), 1 \leq i \leq r$, we can compute $n \in \mathbb{Z} \cap [0, m)$ such that $n \equiv v_i \bmod m$ for $1 \leq i \leq r$ with a cost of $\widetilde{\mathcal{O}}(\log m)$ bit operations.*

Similarly, given n and $m_1, \ldots, m_n$, we can compute $v_i = n \bmod m_i$ for $1 \leq i \leq r$, also with a cost of $\widetilde{\mathcal{O}}(\log m)$ bit operations.

Fast simultenous modular reduction can be achieved via binary splitting or subproduct-tree techniques. Namely, given n, we reduce n modulo $\prod_{i=1}^{\lfloor r/2 \rfloor} m_i$ and $\prod_{i=\lfloor r/2 \rfloor+1}^{r} m_i$ and recurse on each half. Fast Chinese remaindering reconstructs n by traversing this tree of reductions in the reversed order.

Often the methods we use to solve these problems will require us to construct and factor an **auxillary polynomial**, a polynomial whose roots encode information about the support of f. Specifically, over finite fields and integers, we will need to map some product of linear factors $\Phi = \prod_{i=1}^{t}(x - \alpha_i) \in \mathsf{R}[x]$ to $(\alpha_1, \ldots, \alpha_t)$, and vice versa. We can achieve this similarly by a binary splitting method. For a detailed overview of the algorithms and techniques therein, we refer to reader to Section 10 of [VG03]. For our purposes we state the following as a Theorem.

Theorem 2.1.7 ([BM74]). *Given a dense polynomial $\Lambda \in \mathsf{R}[x]$ of degree less than T, and distinct ring elements $\alpha_1, \ldots, \alpha_T \in \mathsf{R}$, we can map Φ to $(\Phi(\alpha_1), \ldots, \Phi(\alpha_T))$ in $\widetilde{\mathcal{O}}(T)$ ring operations. If R is a field we can similarly map $(\Phi(\alpha_1), \ldots, \Phi(\alpha_T))$ to a dense representation of Φ in $\widetilde{\mathcal{O}}(T)$ ring operations.*

2.1.5 Polynomial factorization via Hensel lifting

Given an auxillary polynomial $\Phi \in \mathbb{Z}[y]$ comprised of linear factors, we will need to reconstruct Φ from its linear factors modulo a choice of prime p. We can do this via *Hensel lifting*.

Theorem 2.1.8 (Theorem 15.18, [VG03]). *Let $\Lambda = \prod_{i=1}^{T}(y - \alpha_i) \in \mathbb{Z}[y]$, and suppose we are given the linear factors $(y - \alpha_i) \bmod p$, for some prime p and $i = 1, 2, \ldots, D$. Then for a choice of $\ell \in \mathbb{Z}_{>0}$, we compute $(y - \alpha_i) \bmod p^{\ell}$, for $i = 1, 2, \ldots, T$ with a bit cost of $\widetilde{\mathcal{O}}(T\ell \log^2 p)$.*

In particular, if the factors of Λ have height at most C and we have the factors of Λ modulo a prime $p \in \operatorname{polylog}(T \log C)$ then one can recover Λ in $\widetilde{\mathcal{O}}(T \log C)$ bit operations.

2.1.6 Linear algebra

Much as in the case of integer and polynomial arithmetic, the algorithmic cost of many linear algebra problems may be reduced to multiplication, in this case, the multiplication of matrices. We let $\omega = \omega_{\mathsf{K}}$ denote the exponent of the cost of matrix multiplication, that is, a parameter such that two $n \times n$ matrices over a field K can be multiplied in $\mathcal{O}(n^{\omega+\epsilon})$ K-operations for any $\epsilon > 0$. This parameter ω may depend on, if anything, the characteristic of the underlying field K (see, e.g., Cor. 15.18, [BCS10]). By LeGall [Le 14] we can take $\omega < 2.3728639$ for an arbitrary field K. There is a large gap between the K-operation cost of the least asymptotically expensive algorithms for matrix multiplication, and the best known lower bounds.

In some instances in multivariate linear algebra, we will need to solve integer linear systems. To bound the bit complexity of such operations we cite the following.

Theorem 2.1.9 (Thm. 39, [Sto05])**.** *Let $\mathbf{A}^{m\times n}, m \in \mathcal{O}(n)$, have rank n and $\boldsymbol{b} \in \mathbb{Z}^{m\times 1}$ be given. There exists a Las Vegas algorithm that either:*

- *produces a solution $\boldsymbol{x}$ to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x} \in \mathbb{Q}^n$ is a solution with minimal demonimator (the denominator of $\boldsymbol{x}$ is the least $d \in \mathbb{Z}_{>0}$ such that $d\boldsymbol{x} \in \mathbb{Z}^n$), or*
- *certifies that no such solution exists.*

The expected number of bit operations is $\widetilde{\mathcal{O}}(n^\omega \log(C))$, where C is a bound on the absolute values of the entries of $\boldsymbol{A}$ and $\frac{1}{n}\boldsymbol{b}$.

2.1.6.1 Structured linear systems

We often need to solve linear systems where the accompanying matrix admits special structure. The most common systems of this form that we will need to solve involve **Hankel**, **Toeplitz**, and **Vandermonde** matrices of the respective forms

$$\underbrace{\begin{bmatrix} h_1 & h_2 & \cdots & h_n \\ h_2 & h_3 & \cdots & h_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ h_n & h_{n+1} & \cdots & h_{2n} \end{bmatrix}}_{\stackrel{\text{def}}{=} \mathbb{H}(h_1,\ldots,h_{2n})}, \underbrace{\begin{bmatrix} t_0 & t_1 & \cdots & t_{n-1} \\ t_{-1} & t_0 & \cdots & t_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{-n} & t_{-n+1} & \cdots & t_0 \end{bmatrix}}_{\stackrel{\text{def}}{=} \mathbb{T}(t_{-n},\ldots,t_{n-1})}, \underbrace{\begin{bmatrix} v_1^0 & v_1^1 & \cdots & v_1^{n-1} \\ v_2^0 & v_2^1 & \cdots & v_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_n^0 & v_n^1 & \cdots & v_n^{n-1} \end{bmatrix}}_{\stackrel{\text{def}}{=} \mathbb{V}(v_1,\ldots,v_n)}. \tag{2.2}$$

These and related matrices can easily be represented using an ordered list of $\mathcal{O}(n)$ entries, e.g., we can represent a Vandermonde matrix as $(v_1, v_2, \ldots, v_n)$. For systems involving these and

related structured matrices, an algebraic RAM may solve these systems in deterministic softly-linear time. In [Pan89], Victor Pan unified the analysis of such systems. Pan observed that each of these systems are related to low-rank matrices via linear operators. For a matrix $\boldsymbol{M}$ of any of the above forms, there exists a linear operator L of one of the following forms

$$\nabla_{\boldsymbol{A},\boldsymbol{B}} : \mathsf{K}^{n\times n} \to \mathsf{K}^{n\times n}, \quad \boldsymbol{M} \mapsto \boldsymbol{A}\boldsymbol{M} - \boldsymbol{M}\boldsymbol{B}, \tag{2.3}$$

$$\Delta_{\boldsymbol{A},\boldsymbol{B}} : \mathsf{K}^{n\times n} \to \mathsf{K}^{n\times n}, \quad \boldsymbol{M} \mapsto \boldsymbol{M} - \boldsymbol{A}\boldsymbol{M}\boldsymbol{B}, \tag{2.4}$$

for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ such that $L(\boldsymbol{M})$ has low rank. These operators are known as **displacement operators** of the **Sylvester** and **Stein type** respectively. We say that $\boldsymbol{M}$ has low **displacement rank**. Typically we assume $\boldsymbol{A}$ and $\boldsymbol{B}$ are sparse and with sparse inverses, i.e., such that $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{A}^{-1}, \boldsymbol{B}^{-1}$ have $\mathcal{O}(n)$ nonzero entries. Victor Pan gave a general strategy for solving linear systems with matrices of low displacement rank. We cite the following theorem.

Theorem 2.1.10 ([Pan01])**.** *Let $\boldsymbol{M} \in \mathsf{K}^{n\times n}$ be a Hankel, Toeplitz, or Vandermonde matrix, or a* **Hankel-plus-Toeplitz matrix** *(the sum of a Hankel matrix and a Toeplitz matrix), given in a natural compressed representation. Then if $\boldsymbol{M}\boldsymbol{a} = \boldsymbol{b}$, one can compute $\boldsymbol{a}$ from $\boldsymbol{M}$ and $\boldsymbol{b}$, and similarly $\boldsymbol{a}$ from $\boldsymbol{M}$ and $\boldsymbol{b}$, using $\widetilde{\mathcal{O}}(n)$ field operations.*

These linear systems play a critical role in polynomial interpolation. A Vandermonde system, for instance, gives the relationship between the evaluations and coefficients of a dense polynomial. The Vandermonde case of theorem 2.1.10 is equivalent to Theorem 2.1.7 in the case that R is a field.

2.1.6.2 The transposition principle

We are also would like to solve systems involving the transposes of the matrices of Theorem 2.1.10. To that end we cite the **transposition principle**. The transposition principle roughly states that multiplying a vector by a matrix $\boldsymbol{M}$ is almost the same cost of multiplying by $\boldsymbol{M}^\top$. More specifically, for any algorithm that computes a matrix-vector product $\boldsymbol{M}\boldsymbol{x}$ as a sequence of vector operations, i.e. a sequence of matrix-vector products $\boldsymbol{M}_1(\boldsymbol{M}_2(\ldots(\boldsymbol{M}_k\boldsymbol{x})))$ where multiplication by each matrix $\boldsymbol{M}_i$ is done via the classical row-by-column method (but excluding performing multiplications by 0 and 1 so as to exploit the sparsity of $\boldsymbol{M}$) then $\boldsymbol{M}^\top\boldsymbol{y}$ can be computed as $\boldsymbol{M}_k^\top(\boldsymbol{M}_{k-1}^\top(\ldots(\boldsymbol{M}_1^\top\boldsymbol{y})))$ with roughly comparable cost. Linear solver algorithms over arbitrary fields can generally be expressed in this form.

Theorem 2.1.11 (Transposition principle)**.** *Let $\boldsymbol{M} \in \mathsf{R}^{m\times n}$. If there exists an algorithm running on an algebraic RAM that can compute $\boldsymbol{M}\boldsymbol{x}$ for any vector $\boldsymbol{x} \in \mathsf{R}^n$ via a sequence of vector operations with a cost of u R multiplications and v R additions, then $\boldsymbol{M}^\top\boldsymbol{y}$ can be computed for any vector $\boldsymbol{y} \in \mathsf{R}^m$ in u multiplications and $v + \mathcal{O}(m+n)$ operations.*

In other words, Theorem 2.1.10 holds for the transposes of Hankel, Toeplitz, Vandermonde, and Hankel-plus-Toeplitz matrices as well. The transposition principle has been reinvented multiple times, many of which are documented by Berstein in [Ber]. We follow Bernstein and attribute the transposition principle to Fiduccia (Theorems 4 and 5, p. 112, [Fid73]). The transposition principle is also known as *Tellegen's principle*, after Dutch electrical engineer Bernard D. H. Tellegen.

2.2 Data structures

We will often have to construct dictionaries to store homomorphic images of polynomials. In some of our algorithms we will need to store results in a *dictionary* or an *associative array*.

Definition 2.2.1. *Let $\mathcal{K}$ be a set with a comparison operation $\succ$ (i.e., a total ordering of $\mathcal{K}$). A* **dictionary** *is a collection $\mathcal{D}$ of key-value pairs (k, v), where each key $k \in \mathcal{K}$ is unique. $\mathcal{D}$ supports the following operations*

- $\mathcal{D}$.`add`(k, v)*: adds key-value pair (k, p) to $\mathcal{D}$.*
- $\mathcal{D}$.`remove`(k)*: if $\mathcal{D}$ has a key-value pair of the form (k, v), it removes that key-value pair.*
- $\mathcal{D}$.`lookup`(k)*: if $\mathcal{D}$ has a key-value pair of the form (k, v), it produces v; otherwise it produces* `fail`.
- $\mathcal{D}$.`keys`$()$*: produces a list of all keys k for which $\mathcal{D}$ has a key-value pair (k, v), sorted in increasing order according to $\succ$.*

One could implement such a dictionary in a number of ways, e.g., a hash table or a binary tree. For the purposes of our analysis we will use a red-black tree (see, e.g., Sec. 13, [Cor+09]).

Lemma 2.2.2. *A dictionary $\mathcal{D}$ can be implemented on an algebraic RAM using a red-black tree, where, if $\mathcal{D}$ contains t keys, the dictionary operations cost the following:*

- $\mathcal{D}$.`add`(k, v), $\mathcal{D}$.`remove`(k) *and* $\mathcal{D}$.`lookup`(k)*: $\mathcal{O}(\log t)$ comparisons similarly many bit operations,*
- $\mathcal{D}$.`keys`$()$*: $\widetilde{\mathcal{O}}(t)$ bit operations.*

The space overhead in addition to the key-value pairs is $\widetilde{\mathcal{O}}(t)$ bits.

More generally, we will require multi-dimensional associative arrays whose keys are m-tuples. Moreover, we will need to be able to efficiently search for all values associated with a key $\boldsymbol{k}$ with a given prefix $\boldsymbol{p}$. Specifically, we define the following.

Definition 2.2.3. *Let $\mathcal{K}$ be a set with a comparison operation $\succ$. A* **prefix dictionary** *is a dictionary whose keys are m-tuples $\boldsymbol{k} \in \mathcal{K}^m$. We let a vector $\mathbf{p} \in \mathcal{K}^\ell$, $\ell < [m-1]$ denote a* **prefix** *of key $\boldsymbol{k}$ if $p_i = k_i$ for all $i \in [\ell]$. In addition to the defined dictionary operations, a prefix dictionary supports the following operations:*

- $\mathcal{D}$.`remove`$(\boldsymbol{p})$*: removes all key-value pairs $(\boldsymbol{k}, v)$ where $\boldsymbol{k}$ has $\boldsymbol{p}$ as a prefix.*
- $\mathcal{D}$.`keys`$(\boldsymbol{p})$*: produces a list of all keys of $\mathcal{D}$ with prefix $\boldsymbol{p}$, sorted according to the lexicographical ordering induced by $\succ$.*
- $\mathcal{D}$.`prefixes`(ℓ)*: produces a list of all length-ℓ prefixes, for $\ell < m$, sorted according to lexicographical order.*

One could implement a prefix dictionary using a hierarchy of dictionaries. E.g., If our key set was $\mathbb{Z}^2$, at the top level we would have a dictionary indexed by $\mathbb{Z}$. For each key with prefix $i \in \mathbb{Z}$, we would have a dictionary $\mathcal{D}_i$ indexed by key i. For each key (i, j), $\mathcal{D}_i$ would store a key-value pair (j, v), where v is the value to be associated with (k, v).

Lemma 2.2.4. *Let $\mathcal{D}$ be a prefix dictionary that takes length-m tuples $\boldsymbol{k} \in \mathcal{K}^m$ as keys, where $m \in \mathcal{O}(1)$. Then $\mathcal{D}$ can be implemented using a hierarchy of red-black trees, with dictionary operation costs given as stated in Lemma 2.2.2. If $\boldsymbol{p}$ is a prefix of length-ℓ, and $\mathcal{D}$ has r length-ℓ prefixes and s keys with prefix $\boldsymbol{p}$, then the additional prefix dictionary operations require as follows:*

- $\mathcal{D}$.`remove`$(\boldsymbol{p})$, $\mathcal{D}$.`keys`$(\boldsymbol{p})$*: $\mathcal{O}(\log r)$ comparisons of prefixes.*
- $\mathcal{D}$.`prefixes`(ℓ)*: $\widetilde{\mathcal{O}}(r)$ bit and copy operations.*

2.2.1 Constructing dictionaries of terms

In a number of our algorithms we will have to construct a set of sparse polynomials $f_1, \ldots, f_\ell$ and a dictionary of terms. The key to the dictionary will be something to identify a class of nonzero terms, and the value will be the list of all tuples (i, d) such that the degree-d term of f_i has this property. More concretely, we will consider two types of keys: in Section 3.4, our keys will be $d \bmod p$ for a fixed prime p; in Section 3.5 in Chapter 5, our keys will be the constants, i.e., we will want to store all instances that a term with some coefficient c occurs. Often we will have to do this for vector polynomials. In addition, we will have to construct vector polynomials from a set of polynomials. In order to be free ourselves of these technical considerations later on, we discuss them here. All of these operations amount to resorting terms according to a different ordering.

In order to construct a vector polynomial $\boldsymbol{f} = (f_1, \ldots, f_\ell)$, expressed as $\boldsymbol{f} = \sum_{i=1}^{t} \boldsymbol{c}\boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{R}^\ell[x]$, from the sparse representations of $f_1, \ldots, f_\ell \in \mathsf{R}[x]$, one can use a prefix dictionary. We

could, for instance, do this using a dictionary. For each term $cx^{\boldsymbol{d}}$ of each image f_i, we can store a value c with key $(\boldsymbol{d}, i)$.

Algorithm 1: Constructing a sparse vector polynomial

Input: ℓ sparse n-variate polynomials $f_i = \sum_{j=1}^{t_i} c_{ij} x^{\boldsymbol{e}_{ij}} \in \mathsf{R}[\boldsymbol{x}]$, $i \in [\ell]$
Output: A sparse representation of the vector polynomial $\boldsymbol{f} = (f_1, \ldots, f_\ell)$

$\mathcal{D} \leftarrow$ an empty prefix dictionary that takes keys $(\boldsymbol{d}, i) \in \mathbb{Z}_{\geq 0}^{n+1}$, sorted according to lexicographical order;
for $i \leftarrow 1$ **to** ℓ **do**
  for $j \leftarrow 1$ **to** t_i **do** $\mathcal{D}.\texttt{add}((e_{ij}, i), c)$;
$\boldsymbol{f} \leftarrow 0 \in \mathsf{R}^\ell[x]$, a sparse polynomial;
for $\boldsymbol{d} \in \mathcal{D}.\texttt{prefixes}(1)$ **do**
  $\boldsymbol{c} = (c_1, \ldots, c_\ell) \leftarrow \boldsymbol{0} \in \mathsf{R}^\ell$;
  for $i \in \mathcal{D}.\texttt{keys}(\boldsymbol{d})$ **do**
    $c_i \leftarrow \mathcal{D}.\texttt{lookup}((\boldsymbol{d}, i))$;
  $\boldsymbol{f} \leftarrow \boldsymbol{f} + \boldsymbol{c}x^{\boldsymbol{d}}$;
return $\boldsymbol{f}$;

Algorithm 1 describes how one could construct $\boldsymbol{f}$. If $\boldsymbol{f}$ is T-sparse and $f_1, \ldots, f_\ell$ have a combined m terms, then $\mathcal{D}$ will have at most m entries, and at most T length-1 prefixes. It follows that populating the dictionary will entail $\widetilde{\mathcal{O}}(m)$ comparisons of keys of bit size $n \log D \log \ell$, for a bit cost of $\widetilde{\mathcal{O}}(mn \log D)$. Producing the list of keys with prefix $\boldsymbol{d}$ has cost softly-linear in terms of the number of keys outputted. The m `lookup` operations cost $\mathcal{O}(\log \ell T)$ key comparisons, for a bit cost of $\widetilde{\mathcal{O}}(mn \log D)$.

Similarly, when we want to group terms from a set of images of sparse polynomials $f_1, \ldots, f_\ell$ according to their coefficient, we construct a prefix dictionary of terms, where a term $c\boldsymbol{x}^{\boldsymbol{e}}$ of f_i will be stored the the key (c, i) and value $\boldsymbol{e}$. If we wanted to collect terms according to their exponent modulo an integer p, we could store a term cx^e with the key $((e \bmod p), i, e)$ associated to value c.

In general, we will suppose that if we can test a property of a term with cost softly-linear in the bit size of that term, then we can collect terms from among a set of polynomials $f_1, \ldots, f_\ell$ according to that property in time softly linear in the bit size of those images. More generally, we remark that, for any dictionary $\mathcal{D}$ whereby we only retrieve each value a constant number of times, data access will have cost softly linear in the combined bit size of all key-value pairs ever contained in $\mathcal{D}$.

2.3 Probabilistic inequalities

Most of the algorithms presented in this thesis are *Monte Carlo* probabilistic. We use a few elementary probabilistic inequalities which we will use throughout.

Lemma 2.3.1 (Union bound). *Let A_i be an event occuring with probability p_i, for $1 \leq i \leq n$. Then the probability that at least one of $A_1, \ldots, A_n$ occurs is at most $\sum_{i=1}^{n} p_i$.*

The union bound is also known as **Boole's inequality**, named after logician George Boole.

Lemma 2.3.2 (Markov's inequality). *Let X be a nonnegative random variable. Then for $k > 1$, $\Pr[X > k\mathbb{E}[X]] < 1/k$.*

We give the following as Corollary to Markov's inequality, which will be subsequently used to probabilistic construct sets of primes.

Corollary 2.3.3. *Let $\mathcal{U}$ be a finite set and $\mathcal{S} \subset \mathcal{U}$, where $\#\mathcal{S} \geq 2n$, and let $\delta = \#\mathcal{S}/\#\mathcal{U}$. Then if we choose $m \geq \min\left(\lceil 2n\delta^{-1}\mu^{-1}\rceil, \#\mathcal{U}\right)$ distinct elements $a_1, \ldots a_m$ from $\mathcal{U}$ uniformly at random, the probabilty that $\{a_1, \ldots, a_m\} \cap \mathcal{S} \geq n$ is at least $1 - \mu$.*

Proof. Let $a_1, \ldots, a_{\#\mathcal{U}}$ be an ordering of $\mathcal{U}$, and let X_i be the indicator variable for the event that $a_i \in \mathcal{S}$. Fix i and suppose fewer than n of $a_1, \ldots, a_{i-1}$ are in $\mathcal{S}$. Then the conditional probability that $X_i = 1$ is at least $\delta/2$. It follows that the expected value of the least k such that $a_1, \ldots, a_k$ contains n elements of $\mathcal{S}$ is less than $2n\delta^{-1}$. The result follows from Markov's inequality. □

We give a version of *Hoeffding's inequality* for Bernoulli random variables (i.e. coin flips with a weighted coin).

Theorem 2.3.4 (Hoeffding's inequality; Thm. 1.1, [DP09]**).** *Let $X_1, \ldots, X_n$ be independently distributed in $[0, 1]$, and let $X = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then*

- $\Pr[X < (1-\epsilon)\mathbb{E}[X]], \quad \Pr[X > (1+\epsilon)\mathbb{E}[X]] \quad \leq \quad \exp\left(\frac{-\epsilon^2}{2}\mathbb{E}[X]\right)$;
- $\Pr[X > \mathbb{E}[X] + \epsilon], \quad \Pr[X < \mathbb{E}[X] - \epsilon] \quad \leq \quad e^{-2\epsilon^2 n}$.

Sometimes we will work with random variables that are not purely independent, but such that the "failure" of one random variable only increases the "success" of another.

Definition 2.3.5. *We say a set of random variables $X_i, i \in [n]$ are* **negatively associated** *if for all disjoint subsets $I, J \subset [n]$ and for all nondecreasing functions f and g, $\mathbb{E}[f(X_i, i \in I)g(X_j, j \in J)] \leq \mathbb{E}[f(X_i, i \in \mathcal{I})]\mathbb{E}[g(X_j, j \in J)]$.*

As an example, we prove negative association of a family of sets of random variables that we will subsequently work with.

Lemma 2.3.6. *Suppose we have N bins of which M contain red balls, and the remaining contain white balls. We draw from distinct $n \leq N$ bins, and let X_i be the indicator variable for the event that the ith draw produces a red ball.*

Intuitively if we let $N = n = 2$ and $M = 1$, the variables X_1, X_2 are negatively associated because $X_1 = 1$ forces $X_2 = 0$ and vice versa.

Proof of Lemma 2.3.6. By linearity of expectation, without loss of generality we may translate f and g such that $f(0, \ldots, 0) = 0$ and $g(0, \ldots, 0) = 0$; we also may choose $I = [k], k \leq n$ and $J = [n] \setminus I$, or else replace n with $|I \cup J|$ and relabel the X_i accordingly.

We prove my induction on N. The case $N = 1$ is clear as either f or g is a constant in this case. Now suppose the induction hypothesis is true for all $N \in [N']$ and consider the case $N = N' + 1$. Define

$$E_b = \mathbb{E}[f(b, X_2 \ldots, X_k) g(X_j, j \in J)], \quad E_b^* = \mathbb{E}[f(b, X_2, \ldots, X_k)]\mathbb{E}[g(X_j, j \in J)], \quad b = 0, 1.$$

By the induction hypothesis, $E_b \leq E_b^*$ for $b = 0, 1$. Letting $\mu = M/N$, we have that

$$\begin{aligned} \mathbb{E}[f(X, i \in I) g(X_j, j \in J)] &= (1 - \mu) E_0 + \mu E_1, \\ &\leq (1 - \mu) E_0^* + \mu E_1^* = \mathbb{E}[f(X_i, i \in I)]\mathbb{E}[g(X_j, j \in J)], \end{aligned}$$

completing the proof. □

The following Lemma allows us to use Hoeffding's inequality on negatively associated random variables.

Lemma 2.3.7 (Hoeffding's bound with negative dependence; Thm 3.1, [DP09])**.** *Let $X_1, \ldots, X_n$ be a set of negatively associated random variables taking values in $[0, 1]$. Then the inequalities of Theorem 2.3.4 hold.*

2.3.1 Amplifying probabilistic algorithms

We now describe a standard technique we will frequently use in order to raise the probability of success for a probabilistic algorithm from $2/3$ to $1 - \mu$ for arbitrarily large μ. We call this technique **probabilistic amplification**, and consider two cases of probabilistic algorithms.

Suppose that **UniqueProbProc** is a probabilistic procedure that takes as input a tuple $\boldsymbol{a}$, and, for a fixed choice of $\boldsymbol{a}$, produces a unique correct input with probability at least $1 - c$, for a

constant $c < 1/2$. Then by Hoeffding's inequality, if we run **UniqueProbProc** with input $\boldsymbol{a}$ n times, where n is at least $\ln(\mu^{-1})/(2(c-1/2)^2)$, then **UniqueProbProc**($\boldsymbol{a}$) will produce the correct output less than $n/2$ times with probability at most

$$\exp(-2(c-1/2)^2 n) = \mu^{-1}.$$

By setting n to be odd, it guarantees that the majority output will be the correct output with probability at least $1-\mu$.

Now suppose that **LasVegasProc** is a probabilistic procedure that produces a correct output, not necessarily unique for a fixed input, with probability at least $1-c$, for a constant $c \in [0,1)$, and produces `fail` in the event that it does not produce a correct output. Then the probability that **LasVegasProc** fails $n = \lceil \log_{1/c}(\mu^{-1}) \rceil$ is at most $c^n \leq \mu$. As you may have guessed by its name, **LasVegasProc** can be made Las Vegas probabilistic by merely running it until it produces an output that is not `fail`.

Theorem 2.3.8. *Suppose **ProbProc** is a probabilistic algorithm that takes an input $\boldsymbol{a}$ and satisfies one of the following:*

- ***ProbProc** produces a unique correct solution for a fixed input $\boldsymbol{a}$, and succeeds with probability at least $1-c$ for a constant $c < 1/2$;*
- ***ProbProc** produces `fail` in the event of failure and succeeds with probability at least $1-c$ for a constanct $c < 1$.*

*Then if **ProbProc($\boldsymbol{a}$)** admits a cost of $\mathcal{O}(\Phi(\boldsymbol{a}))$, then we can modify **ProbProc($\boldsymbol{a}$)** to succeed with probability $1-\mu$ for arbitrary $\mu \in (0,1)$ and cost $\mathcal{O}(\Phi(\boldsymbol{a})\log(\mu^{-1}))$.*

Remark 2.3.9. *We establish the following convention. We will write **ProbProc**$(a_1,\ldots,a_m;\mu)$ to denote that **ProbProc**$(a_1,\ldots,a_m;\mu)$ succeeds with probability at least $1-\mu$. Probabilistic algorithms satisfying the criteria of Theorem 2.3.8 will generally be written so as to succeed with probability at least $2/3$. For such a procedure **ProbProc**, when we write **ProbProc**$(a_1,\ldots,a_m;\mu)$, where $\mu < 1/3$, it is implicit that we amplify the probability of success in the manner described in this section.*

2.4 Selecting primes

2.4.1 Constructing the first k primes via sieve methods

We will use sieve methods in order to construct the first k primes for a choice of k.

Theorem 2.4.1 ([Pri82])**.** *One can compute all prime numbers up to $n \in \mathbb{Z}_{>0}$ in $\widetilde{\mathcal{O}}(n)$ bit operations.*

Using the sieve of Eratosthenes requires $\mathcal{O}(n \log\log n)$ additions for a bit-operation cost of $\mathcal{O}(n \log n \log\log n)$; with the wheel sieve this can be reduced to $\mathcal{O}(n/\log\log n)$ additions or $\mathcal{O}(n \log n/\log\log n)$ bit operations [Pri82].

We state the *Prime Number Theorem* for completeness.

Theorem 2.4.2 (prime number theorem (PNT)). $\lim_{k\to\infty} p_k/(k \ln k) = 1$, *where* p_k *is the* k*th least prime number.*

By the PNT, constructing the first k primes requires $\mathcal{O}(k \log^2 k/\log\log\log k)$, or merely $\widetilde{\mathcal{O}}(k)$, bit operations. We state this as a Corollary.

Corollary 2.4.3. *One can construct the first* k *prime numbers in* $\widetilde{\mathcal{O}}(k)$ *bit operations.*

2.4.2 Selecting random primes from a specified interval

Often as a subroutine, we will need to select a prime at random from an interval containing a sufficiently large number of primes. Because we will often consider the set of primes in an interval $(\lambda..2\lambda]$, where $\lambda \in \mathbb{Z}$, we will write

$$\mathcal{P}_{(\lambda..2\lambda]} \stackrel{\text{def}}{=} \{p \in (\lambda..2\lambda] : p \text{ is prime}\}.$$

Throughout Section 2.4 we will also let $\pi : \mathbb{R} \to \mathbb{Z}_{\geq 0}$ denote the **prime counting function**, where

$$\pi(x) \stackrel{\text{def}}{=} \#\{p \leq x : p \text{ is prime}\}.$$

We will rely heavily on the following effective version of the Prime Number Theorem due to Rosser and Schoenfeld.

Lemma 2.4.4 (Cor. 3, [RS62])**.** *We have*

$$\frac{3\lambda}{5\ln\lambda} < \#\mathcal{P}_{(\lambda..2\lambda]} < \frac{7\lambda}{5\ln\lambda}, \quad \textit{for } \lambda \geq 20.5, \quad \textit{and} \quad \frac{\lambda}{\ln\lambda} < \pi(\lambda) < \frac{5\lambda}{4\ln(\lambda)} \quad \textit{for } \lambda \geq 114.$$

Sometimes we will like to choose λ such that $(\lambda..2\lambda]$, for some $k \geq 1$ and bound $B > 1$, $(\lambda..2\lambda]$ contains at least $k \log_\lambda B$ primes, such that any proportion of $1/k$ of the primes in $(\lambda..2\lambda]$ have a product at least B. This is useful when we need to reconstruct some value $v \in \mathbb{Z} \cap [0, B)$ from a set of primes, and we know that we can obtain $v \bmod p$ for a proportion $1/k$ of all primes in $(\lambda..2\lambda]$. In which case, it suffices that

$$\tfrac{3}{5}\lambda \ln^{-1}\lambda \geq k \ln B \ln^{-1}\lambda \quad \iff \quad \lambda \geq \tfrac{5}{3} k \ln(B).$$

We state this as a Corollary.

Corollary 2.4.5. *Let $k, B \in \mathbb{R}_{\geq 1}$ and let $\mathbb{Z} \ni \lambda \geq \max(21, \frac{5}{3} k \ln B)$. Then $\#\mathcal{P}_{(\lambda..2\lambda]} \leq k \log_\lambda B$ primes.*

A natural way to select a prime at random from within a range $(\lambda..2\lambda]$ is to select an arbitrary integer within that range and to test that it is prime using a primality test. To that end we cite the primality test of Agrawal, Kayal, and Saxena (AKS).

Theorem 2.4.6 ([AKS04])**.** *The AKS primality test determines whether $p \in \mathbb{Z}_{>0}$ is prime with runtime* $\mathrm{polylog}(p)$.

In particular, [AKS04] gives a runtime of $\widetilde{\mathcal{O}}(\log^{10.5} p)$. Using the AKS primality test as a subroutine, Procedure **GetPrime** gives a Monte Carlo method to generate a prime $p \in (\lambda..2\lambda]$. The pseudocode warrants some explanation. Either **GetPrime** succeeds and produces a prime $p \in (\lambda..2\lambda]$, or it produces `fail`. In the latter case, we will establish the convention that the procedure that called **GetPrime** will produce `fail` as well, such that any algorithm relying on **GetPrime** as a subroutine will fail if **GetPrime** fails.

Procedure GetPrime(λ)

Input: $\lambda \in \mathbb{Z}$, where $\lambda \geq 21$.
Result: With probability at least $2/3$, outputs a prime $p \in (\lambda..2\lambda]$. Otherwise produces `fail`.

```
m ← ⌈(5 ln 3 / 3) ln λ⌉;
repeat m times times
    p ← integer chosen uniformly at random from (λ..2λ] ∩ ℤ;
    if AKS primality test determines p is prime then return p;
return fail;
```

Lemma 2.4.7. *Procedure **GetPrime** produces a prime $p \in (\lambda..2\lambda] \cap \mathbb{Z}$ with probability at least $2/3$. It admits a runtime of* $\mathrm{polylog}(\lambda)$.

Proof. The cost of **GetPrime** is dominated by the primality tests. Per Theorem 2.4.6, the cost of a single primality test is polylogarithmic in λ, such that the cost of all $\mathcal{O}(\log(\lambda))$ primality tests is also polylogarithmic in λ.

We now prove correctness. By Lemma 2.4.4, the probability that an integer p chosen uniformly and at random from $(\lambda..2\lambda]$ is at least $(3/5) \ln^{-1} \lambda$, such that the probability that none of the chosen p are prime is at most $(1 - (3/5) \ln^{-1} \lambda)^m \leq \exp(-(3/5) \ln^{-1} \lambda)^{(5 \ln 3/3) \ln \lambda} = \exp(-\ln 3) = 1/3$, as desired. □

More generally we would often like to construct a set of primes n primes $\mathcal{P}_{(\lambda..2\lambda]}$. This may be done probabilistically via **GetPrimes**(λ, n).

Procedure GetPrimes(λ, n)

Input: $\lambda \in \mathbb{Z}, \lambda \geq 21$; $n \in \mathbb{Z}_{>0}$.

Output: Produces a set of $\min(n, \#\mathcal{P}_{(\lambda..2\lambda]})$ primes $p \in \mathcal{P}_{(\lambda..2\lambda]}$. Succeeds with probability at least $2/3$, otherwise produces `fail`.

1 $\rho \leftarrow \frac{3}{10}\lambda \ln^{-1} \lambda$;
2 **if** $n > \rho$ **then**
3 Generate $\mathcal{P}_{(\lambda..2\lambda]}$ via the wheel sieve;
4 **if** $n > \#\mathcal{P}_{(\lambda..2\lambda]}$ **then return** $\mathcal{P}_{(\lambda..2\lambda]}$;
5 **else return** a random subset of $\mathcal{P}_{(\lambda..2\lambda]}$, $\mathcal{S}$, of cardinality n;
6 $m \leftarrow \min(\lceil \rho \rceil, \lambda)$;
7 $a_1, \ldots, a_m \leftarrow m$ distinct elements from $(\lambda..2\lambda]$;
8 $\mathcal{S} \leftarrow \{\}$;
9 **for** $i \leftarrow 1$ **to** m **do**
10 **if** *AKS test determines* a_i *is prime* **then**
11 Add a_i to $\mathcal{S}$;
12 **if** $\#\mathcal{S} = n$ **then break**;
13 **if** $\#\mathcal{S} < n$ **then return** `fail`;
14 **else return** $\mathcal{S}$;

Proposition 2.4.8. *GetPrimes(n, λ) is correct with probability at least $2/3$ and admits a cost of $\widetilde{\mathcal{O}}(n \cdot \mathrm{polylog}(\lambda))$.*

Proof. We first prove correctness. By Corollary 2.4.5, $\#\mathcal{P}_{(\lambda..2\lambda]} \geq (3/5)\lambda \ln^{-1} \lambda = 2\rho$. Correctness follows in the cases that $n > \rho$ or $m = \lambda$.

Suppose then that the algorithm sets $m = \lceil \rho \rceil$, such that $(\lambda..2\lambda]$ contains at least m integers and at least $2n$ primes. Then by Corollary 2.3.3, taking $\delta = \#\mathcal{P}_{(\lambda..2\lambda]}/\lambda \geq \bar{\delta} = \frac{3}{5}\ln^{-1}\lambda$, and $\mu = 1/3$, and given that $m = 2n\bar{\delta}^{-1}\mu^{-1} \geq 2n\delta^{-1}\mu^{-1}$, the probability that $a_1, \ldots, a_m$ contains at least n primes is at least $1/3$.

We now analyze the cost. In the case that we generate all primes in $(\lambda..2\lambda]$, by Theorem 2.4.1, this costs $\widetilde{\mathcal{O}}(\lambda)$, which for this case, by line 2, is $\widetilde{\mathcal{O}}(n)$. In the case that we select m primes, then the cost is dominated by the AKS primality tests. These cost $\widetilde{\mathcal{O}}(m \cdot \mathrm{polylog}(\lambda)) = \widetilde{\mathcal{O}}(n \cdot \mathrm{polylog}(\lambda))$ in total. □

On some occasions we will need that $(\lambda..2\lambda]$ contains at least n primes for a fixed n. To that

end we note that it suffices, for $\lambda \geq 21$, that

$$\frac{\lambda}{\ln \lambda} \geq \frac{5}{3} n.$$

We note that the solutions to $x \ln^{-1}(x) = c, c \in \mathbb{R}_{\geq -1}$, are

$$x = \exp(W_{-1}(-1/c)), \quad x = \exp(W_0(-1/c)),$$

where $W_{-1}, W_0 : \mathbb{C} \mapsto \mathbb{C}$ are the lower and upper branches of the **Lambert W function**, i.e., the inverse function to $x \mapsto xe^x$ (see, e.g., [Cor+96]). In order to simplify our analysis we would like a solution of the form $\lambda = an \ln(n)$ for an appropriate constant a. We note that, for such a λ,

$$\frac{\lambda}{\ln \lambda} = \frac{an \ln(n)}{\ln(an \ln(n))}.$$

Solving for

$$a \ln(n) = \ln(a) + \ln(n) + \ln(\ln(n))$$

where $n = 21 \ln^{-1}(21)$ gives

$$a = \frac{\ln(21)}{\ln n} \approx 1.576510209 < \frac{8}{5}, \quad a = \frac{W_0(-n^{-1})}{\ln n} \approx 0.08918131506,$$

We note that $\lambda \ln^{-1} \lambda$ is an increasing function for $\lambda \geq e$, and $a \ln n / (\ln(an \ln(n)))$ is an increasing function, with respect to n, for $a = \frac{8}{5}$ and $n \geq 21 \ln^{-1}(21)$. We conclude with the following proposition:

Proposition 2.4.9. *Let $n, \lambda \in \mathbb{Z}_{>0}$ with $\lambda \geq \max(21, \frac{8}{5} n \ln n)$. Then $(\lambda..2\lambda]$ contains at least n primes.*

In some cases we will want to efficiently select a large prime $p \in \widetilde{\Theta}(\lambda)$, i.e., with special attention to the cost with respect to $\log \lambda$. In this setting, we may use, in place of the AKS test, the **Miller–Rabin test** [Mil75; Rab80], which always accepts if p is prime and rejects with probability $1 - \mu$ if p is not prime, with cost $\widetilde{\mathcal{O}}(\log^2 D \log(\mu^{-1}))$ using FFT-based arithmetic. As $p \in \widetilde{\Theta}(\lambda)$ is prime with probability $\widetilde{\Theta}(\log^{-1} \lambda)$, we can modify **GetPrime** and **GetPrimes** in order to find primes with cost $\widetilde{\mathcal{O}}(\log^3 \lambda)$. We state this as a lemma.

Lemma 2.4.10 (Construction of primes via Miller–Rabin [Mil75; Rab80])**.** *There exists an algorithm which finds a prime $p \in (\lambda..2\lambda]$ with probability at least $2/3$ and cost $\widetilde{\mathcal{O}}(\log^3 \lambda)$.*

There exists an algorithm that takes n and λ, and produces $\ell = \max(n, \#\mathcal{P}_{(\lambda..2\lambda]})$ distinct primes $p_1, \ldots, p_\ell \in (\lambda..2\lambda]$ with cost $\widetilde{\mathcal{O}}(n \log^3 \lambda)$

2.4.3 Constructing elements in $\mathbb{Z}_q$ of specified order

In some instances we will need to identify, for a prime q, an $\omega \in \mathbb{Z}_q$ of a specific multiplicative order $d|(q-1)$. We can "guess" $\omega = \zeta^{(q-1)/d}$ for a generator ζ of $\mathbb{Z}_q^*$. We can then test if ω has multiplicative order d by checking that $\omega^{d/p} \neq 1$ for each p dividing d. As ZZ_q^* has $\phi(q-1)$ generators, where ϕ is Euler's totient function, and $\phi(n) \in \mathcal{O}(n/\log\log n)$ (Thm. 328, [Har+08]), we can probabilistically find a generator ζ by selecting $\mathcal{O}(\log\log q)$ $\zeta \in \mathbb{Z}_q^*$ independently and uniformly at random. As d has at most $\mathcal{O}(\log q)$ divisors, this gives us the following:

Lemma 2.4.11. *Given a prime q and some $d > 0$ dividing $q-1$, there exists an algorithm which discovers some $\omega \in \mathbb{Z}_q$ with multiplicative order d with cost $\widetilde{\mathcal{O}}(\log^3 q)$ bit operations. If d is specifically a product of a constant number of prime powers, then we can discover $\omega \in \mathbb{Z}_q$ of multiplicative order d with cost $\widetilde{\mathcal{O}}(\log^2 D)$.*

2.4.4 Selecting primes in arithmetic progressions

In some applications we will need to select a prime q such that $\mathbb{Z}_q$ contains pth roots of unity for an appropriate prime p. In other words, we need a prime q of the form $q = kp+1$ for some $k \in \mathbb{Z}_{>0}$. To that end we define the *prime-counting function over arithmetic progressions*,

$$\pi(y; m, a) = \#\{p \leq y : p \equiv a \bmod m, p \text{ is prime}\}.$$

We cite the following result on the number of primes in arithmetic progressions, a corollary of an effective version of the Bombieri-Vinogradov theorem, due to Akbary and Hambrook.

Lemma 2.4.12 (Corollary 1.4, [AH15])**.** *Let $\gamma \geq 4$, $1 \leq \lambda_1 \leq \lambda_2 \leq \gamma^{1/2}$. Let $\ell(m)$ denote the least prime divisor of m and ϕ be Euler's totient function. Then*

$$\sum_{\substack{m \leq \lambda_2 \\ \ell(m) > \lambda_1}} \max_{2 \leq y \leq \gamma} \max_{\substack{a \\ \gcd(a,m)=1}} \left| \pi(y; m, a) - \frac{\pi(y)}{\phi(m)} \right| < 346.21 \left(4\frac{\gamma}{\lambda_1} + 4\gamma^{1/2}\lambda_2 + 18\gamma^{2/3}\lambda_2^{1/2} + 5\gamma^{5/6}\ln(e\lambda_2/\lambda_1) \right) (\ln \gamma)^{9/2}.$$

Taking $\lambda_1 = \lambda \geq 10^6$, $\lambda_2 = 2\lambda$, and $\gamma = \lambda^3$, gives us the following

Corollary 2.4.13. *Let $\lambda \geq 10^6$, then*

$$\sum_{p \in \mathcal{P}_{(\lambda..2\lambda]}} \left| \pi(\lambda^3; p, 1) - \frac{\pi(\lambda^3)}{p-1} \right| < 48572 \left(4\lambda^2 + (27 + 5\ln(2))\lambda^{5/2} \right) (\ln \lambda)^{9/2}$$
$$\leq 1.48 \cdot 10^6 \lambda^{5/2} \ln^{9/2} \lambda.$$

By Lemma 2.4.4, we have that $\pi(\lambda^3) \geq \lambda^3/(3\ln\lambda)$, such that the number of primes $p \in \mathcal{P}_{(\lambda..2\lambda]}$ for which $\pi(\lambda^3; p, 1) \leq \lambda^2/(6\ln\lambda)$ is at most

$$\frac{1.48 \cdot 10^6 \lambda^{5/2} \ln^{9/2}\lambda}{\frac{1}{6}\lambda^2 \ln^{-1}\lambda} = 8.88 \cdot 10^6 \lambda^{1/2} \ln^{11/2}\lambda.$$

Now, the number of primes in λ is at least $3\lambda/(5\ln\lambda)$, such that the proportion of such primes for which $\pi(\lambda^3; p, 1) < \lambda^2/(6\ln\lambda)$ is at most

$$1.48 \cdot 10^7 \lambda^{-1/2} \ln^{13/2}\lambda. \tag{2.5}$$

If we take $\lambda \geq 2^{191}$, then we can bound (2.5) by $1.64 \cdot 10^{-8}$.

Corollary 2.4.14. *Suppose $\lambda \geq 2^{191}$. Then for a prime p chosen at random from $(\lambda..2\lambda]$, we have that $\pi(\lambda^3; p, 1) \geq \lambda^2/(6\ln\lambda)$ with probability greater than $1 - 1.64 \cdot 10^{-8}$.*

We remark that increasing the probability of failure does not significantly decrease bounds on $\log\lambda$. If we wanted instead that the proportion (2.5) were at most $1/2$ as opposed to $1.64 \cdot 10^{-8}$, we would require that $\log\lambda \geq 155$.

We present two procedures for producing primes in arithmetic progressions. These are both used in Chapter 4, where we give a procedure for the multiplication of sparse polynomials over the integers. Our first procedure, **GetPrimeAP-5/6** is somewhat technical. **GetPrimeAP-5/6**(λ, B) probabilistically produces a prime $p \in (\lambda..2\lambda]$ and a prime $q = ap + 1 \in (\lambda..\lambda^3]$ such that q does not divide an unknown nonzero integer of absolute value at most B.

Lemma 2.4.15. *Let $\lambda, B \in \mathbb{Z}_{>0}$ with $\lambda \geq \max(2^{191}, \sqrt{140\ln B})$. Let b be a fixed, unknown integer with absolute value at most b. With probability at least $5/6$, **GetPrimeAP-5/6**$(\lambda, B; 1/6)$ produces a triple (p, q, ω), where*

- *$p \in (\lambda..2\lambda]$ is a prime selected uniformly at random;*
- *$q \in (2\lambda^2..\lambda^3]$ is a prime such that $q \equiv 1 \pmod{p}$ and q does not divide b;*
- *ω is a pth primitive root of unity modulo q.*

*In the case of failure **GetPrimeAP-5/6** either produces the error code* `fail`*, or a triple (p, q, ω) where q divides the unknown integer B. Its cost is $\widetilde{\mathcal{O}}(\mathrm{polylog}(\lambda) + \mathrm{polylog}(\log B))$.*

Proof. We first prove the probabilistic correctness of **GetPrimeAP-5/6**. By the union bound, and Corollary 2.4.14, **GetPrime** produces a prime in $p \in (\lambda..2\lambda]$ for which there are at least $\lambda^2/(6\ln\lambda)$ primes $q \leq \lambda^3$ of the form $q = aq + 1$ with probability at least $1 - \frac{1}{15} - 1.64 \cdot 10^{-8}$. Assume this is

Procedure GetPrimeAP-5/6($\lambda, B; 5/6$)

Input: $\lambda \in \mathbb{Z}$, where $\lambda \geq 2^{191}, \sqrt{140 \ln B}$.

Output: With probability exceeding $5/6$, a triple (p, q, w) where $p \in (\lambda..2\lambda]$ is prime, $q \in (2\lambda^2..\lambda^3]$ is a prime of the form $q = ap + 1$ and such that q does not divide a fixed integer $B \in \mathbb{Z}_{>0}$, and ω is a pth root of unity in $\mathbb{Z}_q$.

```
p ← GetPrime(λ; 1/20);
(ℓ1, ℓ2) ← ⌊(2λ − 1)/p⌋, ⌊(λ^3 − 1)/p⌋;
m ← ⌈140 ln λ⌉;
repeat m times
    Choose a ∈ (ℓ1..ℓ2] uniformly at random;
    q ← ap + 1;
    if q is prime then
        Choose ζ ∈ Z_q^* uniformly at random;
        e ← (q − 1)/p ∈ Z;
        ω ← ζ^e;
        if ω^p mod q ≠ 1 then return (p, q, ω);
return fail;
```

the case. As $ap+1$ is not prime for $a = 0$ and odd a, there are at most λ primes of the form $ap+1$, $a \in [\ell_1]$ such that there are at least $\lambda^2/(6 \ln \lambda) - \lambda$ primes of the form $ap + 1$, where $a \in (\ell_1..\ell_2]$. For $\lambda \geq 2^{191}$, we have

$$\lambda^2/(6 \ln \lambda) - \lambda > \lambda^2/(7 \ln \lambda).$$

Thus probability that $ap + 1$ is prime for $a \in (\ell_1..\ell_2]$ chosen uniformly at random is more than $\lambda^2/(7\ell_2 \ln \lambda) \geq 1/(7 \ln \lambda)$. Thus, over $m \geq 140 \ln \lambda$ iterations, the expected number of primes q that would be produced is at least 20, such that by Markov's inequality, at least one iteration of the loop produces a prime q with probability at least $1 - 1/20$.

We will say a prime q is "bad" if it divides our unknown nonzero integer with absolute value at most B. As there are at most $\log_\lambda B$ bad primes q that divide our fixed nonzero integer. By our assumption and the choice of $\lambda \geq \sqrt{140 \ln B}$, there are at least $\lambda^2/(7 \ln \lambda) \leq 20 \log_\lambda B$ primes q of the form $q = ap + 1, q \leq \lambda^3$. Thus the probability that such a q chosen uniformly at random is a bad prime is at most $1/20$.

The probability that ω is not a primitive pth root of unity is at least $1/p \geq 2^{-191}$, as $p \geq \lambda \geq 2^{-191}$. Thus by the union bound, the entire procedure fails with probability at most $3/20 + 2^{-191} + 1.64 \cdot 10^-8 \leq 1/6$.

The cost is dominated by primality testing. The cost of m primality tests is $\widetilde{\mathcal{O}}(m \cdot \mathrm{polylog}(\lambda)) = \widetilde{\mathcal{O}}(\mathrm{polylog}(\lambda))$. By the choice of $\lambda \in \Omega(\sqrt{\log B})$, this cost is $\widetilde{\Omega}(\mathrm{polylog}(\log B))$. □

We remark that it is difficult to construct **GetPrimeAP-5/6** to succeed with arbitrarily large probability. This is because we cannot efficiently detect that case that p is a prime for which there are not many primes of the form $q = ap + 1$, which would then affect the probability that such a q does not divide our unknown integer. However, to fit **GetPrimeAP-5/6** within other calling procedures with other probabilistic steps, we designed it to succeed with probability at least $5/6$ we name our procedure **GetPrimeAP-5/6** to remind the reader of this.[2]

GetPrimesAP(λ, C) probabilistically produces a prime $p \in (\lambda..2\lambda]$ and a list of tuples of the form (ω_i, q_i) where the $q_i \in (\lambda..\lambda^3]$ are distinct primes of the form $q_i = ap + 1$, $\omega_i \in \mathbb{Z}_q$ is a pth primitive root of unity, and there are sufficiently many q_i such that one can reconstruct an integer $k \in [0..C]$ from its congruences modulo the q_i. Unlike **GetPrimeAP-5/6**, we can detect when failure occurs, such that **GetPrimeAP-5/6** can be made to succeed with arbitrary probability via *probabilistic amplification*.

Lemma 2.4.16. *Suppose $\lambda \geq \max(2^{191}, \sqrt{42 \ln C})$. Then **GetPrimesAP**$(\lambda, C)$ probabilistically produces a prime $p \in (\lambda, 2\lambda]$ and $k = \lceil \log_\lambda C \rceil$ pairs (q_i, ω_i) where each $q_i = a_i p + 1 \leq \lambda^3$ is prime and ω_i is a pth root of unity. It succeeds with probability at least $2/3$. In the case of failure it produces* `fail`. *Its cost is $\widetilde{\mathcal{O}}(m \cdot \mathrm{polylog}(\lambda))$.*

Proof. First, by the choice of λ we have that, for p chosen uniformly at random from $\mathcal{P}_{(\lambda,2\lambda]}$, that with probability at least $1 - 1.64 \cdot 10^{-8}$, there are at least $\lambda^2/(6 \ln \lambda) \geq 7 \log_\lambda C$ primes of the form $q = ap + 1$, $a \in [\ell]$. Suppose this is the case, such that the probability that $ap + 1$ is prime for a randomly selected $a \in [1..\ell]$ exceeds $1/(6 \ln \lambda)$. If $m = \ell$ then we necessarily will construct at least $7 \log_\lambda C$ primes. The expected number of primes amongst values $a_i p + 1, i \in [m]$, exceeds $m/(6 \ln \lambda) \geq \log_\lambda C$. Thus by Markov's inequality, this exceeds $7 \log_\lambda C$ with probability at least $1/7$.

Note that, for a prime $q = ap + 1$ and $\omega \in \mathbb{Z}_q^*$, that ω^a is either a pth root of unity, or 1. Moreover, there are precisely a $\omega \in \mathbb{Z}_q^*$ such that $\omega^a = 1$, such that the probability that ω selected uniformly at random satisfies $\omega^a \neq 1$ is at least $1 - 1/p \geq 1 - 2^{-191}$. Thus the expected number of pairs $(q_i, \omega_i) \in \mathcal{L}$ is at least $7(1 - 2^{192}) \log_\lambda C$, such that $\#\mathcal{L} \geq \log_\lambda C$ with probability at least $1/(7(1 - 2^{192}))$. Thus, by the union bound the procedure succeeds with probability at least

$$1 - 1.64 \cdot 10^{-8} - 1/7 - \frac{1}{7(1 - 2^{192})} \approx 0.71429 > 2/3.$$

We now analyze the cost. The cost of the call to GetPrime is $\widetilde{\mathcal{O}}(\mathrm{polylog}(\lambda))$. The cost of the m primality tests on the q_i is $\widetilde{\mathcal{O}}(m \cdot \mathrm{polylog}(\lambda)) \subseteq \widetilde{\mathcal{O}}((\log C) \cdot \mathrm{polylog}(\lambda))$. The cost of computing

[2] More realistically, to remind the author.

Procedure GetPrimesAP(λ, C)

Input: $\lambda \in \mathbb{Z}, \lambda \geq 2^{191}, \sqrt{42 \ln C}$; $C \in \mathbb{Z}_{>0}$

Output: With probability exceeding $2/3$, a prime $p \in (\lambda, 2\lambda]$ and a list $\mathcal{L}$ of $\log_\lambda C$ pairs (q, ω) such that the q are distinct primes of the form $q = ap + 1, q \leq \lambda^3$, and $\omega \in \mathbb{Z}_q$ is a pth root of unity.

$p \leftarrow$ **GetPrime**$(\lambda; 1/16)$;
$\ell \leftarrow \lfloor (\lambda^3 - 1)/p \rfloor$ $m \leftarrow \min(\ell, 84 \ln C)$;
Select distinct $a_1, \ldots, a_m \in [\ell]$;
$\mathcal{Q} \leftarrow \{a_i p + 1 : i \in [m]\}$;
if $\mathcal{Q}$ *contains fewer than* $7 \log_\lambda C$ *primes* **then**
    return `fail`
$q_1, \ldots, q_k \leftarrow$ primes in $\mathcal{Q}$;
for $i \leftarrow 1$ **to** k **do**
    Choose $\omega \in \mathbb{Z}_{q_i}^*$ uniformly at random, for $i \in [\ell]$;
    $a = (q_i - 1)/p \in \mathbb{Z}$;
    $\omega_i = \omega^a$;
$\mathcal{L} \leftarrow \{(q_i, \omega_i \bmod q_i) : \omega^{a_i} \neq 1 \bmod q_i, i \in [\ell]\}$;
if $\#\mathcal{L} \leq \log_\lambda C$ **then return** `fail`;
return $p, \mathcal{L}$;

$\omega^a \bmod q_i$ via square-and-multiply is $\widetilde{\mathcal{O}}(\log a \log q_i) \subseteq \widetilde{\mathcal{O}}(\mathrm{polylog}(\lambda))$. Doing this for at most m q_i costs $\widetilde{\mathcal{O}}(m \cdot \mathrm{polylog}(\lambda)) = \widetilde{\mathcal{O}}((\log C) \cdot \mathrm{polylog}(\lambda))$. □

2.5 Probabilistic black-box polynomial identity testing

In this section we present results that will allow us to probabilistically test whether an unknown polynomial f is zero, given a bound on the degree of f and the ability to evaluate f. We first cite a lemma due to Schwartz, and include his concise proof.

Theorem 2.5.1 (Lem. 1, [Sch80]). *Let* Z *be an integral domain and* $f \in \mathsf{Z}[x_1, \ldots, x_n]$ *be an* n*-variate nonzero polynomial. Let* $d_1 = \deg_{x_1}(f)$*, and let* $f_1 \in \mathsf{K}[x_2, \ldots, x_n]$ *be the coefficient of the* $x_1^{d_1}$ *term of* f*, treating* f *as an element in* $\mathsf{K}[x_2, \ldots, x_n][x_1]$*. Similarly define* $d_2 = \deg_{x_2}(f_1)$ *and let* f_2 *be the coefficient of the* $x_2^{d_2}$ *of* f_1*. Recursively define* $f_1, \ldots, f_{n-1}$ *and* $d_1, \ldots, d_n$ *in this fashion. Then for finite sets* $\mathcal{S}_1, \ldots, \mathcal{S}_n \subseteq \mathsf{Z}$*, and for* $\boldsymbol{\omega}$ *chosen uniformly at random from the Cartesian*

product $\prod_{i=1}^{n} \mathcal{S}_i$*,* $f(\boldsymbol{\omega}) \neq 0$ *with probabilty at least*

$$1 - \sum_{i=1}^{n} \frac{d_i}{\#\mathcal{S}_i}.$$

Proof. We prove by induction on n, the number of variables. For the case that $n = 1$, we note that the number of roots of $f \in \mathsf{Z}[x_1]$ is at most its degree (see, e.g., Chap. IV, Sec. 1, Thm. 1.1, [Lan05]).

Now suppose the theorem holds for any k-variate nonzero polynomial over Z, $k < n$. If $f(\boldsymbol{\omega}) = 0$, then it holds that either $f_1(\omega_2, \ldots, \omega_n) = 0$, or $f^*(\omega_1) = 0$, where $f^* = f(x_1, \omega_2, \ldots, \omega_n)\mathsf{Z}[x_1]$. Then by the induction hypothesis, $f_1(\omega_2, \ldots, \omega_n) = 0$ with probability at most $\sum_{i=2}^{n} d_i/\#\mathcal{S}_i$, and $f^*(x_1) = 0$ with probability at most $d_1/\#\mathcal{S}_1$. Taking the union bound for these two events completes the proof. □

As the total degree of f is at most $\sum_{i=1}^{n} d_i$, taking $\mathcal{S}_1 = \mathcal{S}_2 = \cdots = \mathcal{S}_n$ gives the following corollary, which we state as a theorem.

Theorem 2.5.2 (DeMillo–Lipton–Schwartz–Zippel Lemma)**.** *Let* Z *be a field and* $f \in \mathsf{Z}[x_1, \ldots, x_n]$ *be a nonzero polynomial with total degree* D^**. Let* $\mathcal{S} \subset \mathsf{Z}$*, and choose* $\boldsymbol{\omega} \in \mathcal{S}^n$ *uniformly at random. Then* $f(\boldsymbol{\omega}) \neq 0$ *with probability at least* $1 - D^*/\#\mathcal{S}$

Theorem 2.5.2 is known as the **Schwartz–Zippel Lemma** or the **DeMillo–Lipton–Schwartz–Zippel Lemma** in literature, due to related work by DeMillo and Lipton, and by Zippel in polynomial identity testing. We will refer to it as the **DLSZ Lemma**.

Thus, for a black-box polynomial f over an integral domain containing in excess of D^* elements by some constant factor exceeding 1, we may probabilistically test that f is zero with a probability of failure at most $\mu > 0$, at a cost of $\widetilde{\mathcal{O}}(\log \mu^{-1})$ black-box queries. Moreover, this test can only be incorrect in the event that f is nonzero. In the case that f has partial degrees at most D, the DLSZ Lemma gives the following corollary:

Corollary 2.5.3. *If* $f \in \mathsf{K}[\boldsymbol{x}]$ *is nonzero and* n*-variate with partial degrees at most* D*, then for any finite set* $\mathcal{S} \subset \mathsf{K}$ *and* $\boldsymbol{\omega} \in \mathcal{S}$ *chosen uniformly at random,* $f(\boldsymbol{\omega}) \neq 0$ *with probability at least* $1 - nD/\#\mathcal{S}$*.*

Zippel used Theorem 2.5.2 for the purposes of probabilistic interpolation. Namely, Zippel's interpolation algorithm chooses random evaluation points for which, with high probability, a set of unknown nonzero polynomials of bounded degree would all evaluate to nonzero values (see Section 5.2.1). Zippel calls this the **zero avoidance problem** [Zip90]. We use the DLSZ Lemma in a slightly different context. We use it to pick evaluation points $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_m$ such that with high

probability, and for an unknown set of polynomials $\mathcal{F}$, for every $f, g \in \mathcal{F}$, $f \neq g$, there exists an $\boldsymbol{\omega}_j$ such that $f(\boldsymbol{\omega}_j) \neq g(\boldsymbol{\omega}_j)$. We do this in a technique called *diversification*, first employed by Giesbrecht and Roche in [GR11] and subsequently generalized to a number of contexts described herein.

The DLSZ Lemma has a lengthy history and many variants exist, many of which were discovered independently multiple times. We refer to [Bis+15] for a detailed history of the Lemma. To the surprise of the author, the result attributed to Zippel does not imply Theorem 2.5.2. Both DeMillo and Lipton in [DL78] and Zippel in [Zip79] prove the following.

Theorem 2.5.4 (DeMillo–Lipton–Zippel Theorem, [DL78] [Zip79])**.** *Let* R *be an integral domain and* $\mathcal{S} \in \mathsf{R}$ *a finite set, and let* $f \in \mathsf{R}[\boldsymbol{x}]$ *be an* n*-variate nonzero polynomial with partial degrees at most* D*. Then for* $\boldsymbol{\omega}$ *chosen independently and uniformly at random,* $f(\boldsymbol{\omega}) \neq 0$ *with probability at least* $(\#\mathcal{S} - d)^n$*.*

A natural question is whether these results extend $f \in \mathsf{R}[\boldsymbol{x}]$ when R has zero divisors? It is easy to construct counterexamples to the DLSZ Lemma in this setting.

Example 2.5.5. *Let* $f(x) = 2^{k-1}x \in \mathbb{Z}_{2^k}[\boldsymbol{x}]$*. Then* $f(2j+1) = 0$ *for all* $j \in [2^k]$*. I.e.,* f *is of degree* 1 *but with* 2^k *roots.*

In [Bis+15], Bishnoi et al. generalize the DLSZ Lemma to arbitrary commutative rings, with identity with additional constraints on the set $\mathcal{S}$.

Definition 2.5.6. *A set* $\mathcal{S} \subset \mathsf{R}$ *is said to satisfy the* regular-difference condition *if, for all* $a \neq b \in \mathcal{S}$*,* $a - b$ *is not a zero divisor (i.e.,* $a - b$ *is* regular*). We will call any set satisfying the regular-difference condition a* **regular-difference set**.

In [Cla14] and [Bis+15], the regular-difference condition is referred to as *Condition (D)*. This condition was previously introduced in [Sch+08], towards a generalization of the Combinatorial Nullstellensatz.

Theorem 2.5.7 (Generalized DLSZ Lemma; Thm. 4.2, 4.3[Bis+15])**.** *Theorems 2.5.1 and 2.5.2 and Corollary 2.5.3 hold with* R *instead a commutative ring with identity, not necessarily a domain, provided* $\mathcal{S}$ *and* $\mathcal{S}_1, \ldots, \mathcal{S}_n$ *are regular-difference sets.*

The Generalized DLSZ Lemma follows from a generalization of the Alon–F uredi Theorem due to Bishnoi et al., which gives an lower bound on the number of nonzero evaluations of a polynomial f over a multidimensional grid $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \mathcal{S}_n$, where each $\mathcal{S}_i \subset \mathsf{R}$. The Generalized Alon-F uredi Theorem is also used to prove a generalization of the DeMillo–Lipton–Zippel Theorem. We may let the "DLSZ Lemma" refer to either Theorem 2.5.2 or 2.5.7.

The probabilistic correctness of many of our algorithms for f over $\mathbb{F}_q$ will rely on the DLSZ Lemma. As such, these algorithms may be readily adapted to arbitrary coefficient rings containing a regular-difference set $\mathcal{S}$ of cardinality at least $(1+\epsilon)D$, ($(1+\epsilon)nD$ in the n-variate case) for some constant $\epsilon > 0$.

2.6 Kronecker Substitution

In this section we introduce the well-known technique of **Kronecker substitution**, which reduces multivariate interpolation to univariate interpolation.

In [Kro82], Leonard Kronecker gave a method for interpolating a black-box polynomial over the integers. Given a black-box polynomial $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathbb{Z}[\boldsymbol{x}]$, if one knows a bound D on the partial degrees of f and $C \geq |f|_\infty = \max_{i=1}^{t} |c_i|$, one can choose $B \geq (2D+1)$, and evaluate

$$a = f(B, B^{(D+1)}, B^{(D+1)^2}, \ldots, B^{(D+1)^{n-1}}) = \sum_{i=1}^{t} c_i B^{\left(\sum_{j=1}^{n} e_{ij}(D+1)^{j-1}\right)} \in \mathbb{Z}.$$

If we write a as a base-B integer in *balanced form*, i.e., as $a = \sum_{j=1}^{\ell} a_j B^j$, with $-B/2 < a_j \leq B/2$, then the B^k digit of a, where $k = \sum_{j=1}^{n} e_i j (D+1)^j$, is exactly the coefficient c_i.

Example 2.6.1. *Suppose we know $f \in \mathbb{Z}[x, y]$ with partial degrees at most $D = 2$ and coefficients at most 3 in magnitude If we choose $B = 8$, then $a = f(8, 8^3) = 98302 = 2 \cdot 8^5 + 7 \cdot 8^4 + 7 \cdot 8^3 + 7 \cdot 8^2 + 7 \cdot 8 + 6$. Converting to balanced form, we get $a = 3 \cdot 8^5 - 2$. As the $8^{(0,0)\cdot(1,D+1)}$ base-8 digit of a is -2, we get that f has a term $-2x^0y^0$. Similarly, as the $8^{(2,1)\cdot(1,D+1)}$ is 3, we ascertain that f has the term $3x^2y$.*

This technique, which we call **integer Kronecker substitution** reduces multiplication of multivariate integer polynomials to multiplication of integers; given inputs $f_1, f_2 \in \mathbb{Z}[\boldsymbol{x}]$, one can ascertain appropriate bounds C and D for the product $f_1 f_2$, such that we can compute the integer product

$$f_1(B, B^(D+1), B^{(D+1)^2}, \ldots, B^{(D+1)^n}) f_2(B, B^{(D+1)}, \ldots, B^{(D+1)^n})$$

from which we can read the coefficients of the product $f_1 f_2$.

More generally, for polynomials over arbitrary rings, one may use a univariate interpolation algorithm of their choosing to interpolate the univariate image of f,

$$g(z) = f(z, z^{(D+1)}, z^{(D+1)^2}, \ldots, z^{(D+1)^{n-1}}) \in \mathsf{R}[z], \tag{2.6}$$

from which we can recover f. We will call this technique **Kronecker substitution**, resulting image g a **substitution** of f. The degree of this univariate image is at most $\sum_{j=1}^{n} D(D+1)^{j-1} = (D+1)^n - 1$. This map has the useful property that an n-variate term $c\boldsymbol{x}^{\boldsymbol{e}}$ of f with partial degrees at most D uniquely maps to a term $cz^{\boldsymbol{e}\cdot\boldsymbol{d}}$, where $\boldsymbol{d} = (1, D+1, (D+1)^2, \ldots, (D+1)^{n-1})$.

More generally, if f has partial degree bounds $D_i \geq \deg(f_i), i \in [n]$, then we may make the substitution

$$g(z) = f(z, z^{D+1}, z^{(D_1+1)(D_2+1)}, \ldots, z^{(D_1+1)\cdots(D_{n-1}+1)}), \tag{2.7}$$

then a term cx^e of f will uniquely map to a term in g of degree $d = \sum_{i=1}^{n}(e_i \prod_{j<i}(D_j+1))$. The partial exponents may be obtained as

$$\begin{aligned}
e_1 &= d \text{ rem } (D_1+1)), \\
e_2 &= (d \text{ quo } (D_1+1)) \text{ rem } (D_2+1), \\
e_3 &= (d \text{ quo } (D_1+1)(D_2+1)) \text{ rem } (D_3+1), \\
&\vdots \\
e_n &= (d \text{ quo } \prod_{j=1}^{n-1}(D_j+1)) \text{ rem } (D_n+1).
\end{aligned} \tag{2.8}$$

These techiques easily extend to multivariate Laurent polynomials. Given a bound D on the partial *absolute degrees* of f, we may instead replace $D+1$ in (2.6) with $(2D+1)$. The partial exponents of an exponent e can be obtained by the base-$(2D+1)$ expansion of e in balanced form. If instead we have bounds D_i on the absolute value of the x_i-degree of f_i, we may replace D_j+1 with $(2D_j+1)$. Instead of taking a positive remainder as in (2.8), we would take the *least absolute remainders*. For $a, b \in \mathbb{Z}, b > 0$, the integer the **least absolute remainder** of a divided by b is $r \in (\lfloor -b/2 \rfloor .. \lfloor b/2 \rfloor]$, and $r \equiv a \pmod{b}$.

A different strategy is taken in [GLL09], whereby one chooses distinct primes $p_1, \ldots, p_n \geq D$, and, for $m = \prod_{i=1}^{n} p_i$, constructs the univariate image

$$g(z) = f(z^{m/p_1}, \ldots, z^{m/p_n}).$$

Here a term with exponent $\boldsymbol{e}$ maps to $a = \boldsymbol{e} \cdot \boldsymbol{p}$, where $\boldsymbol{p} = (m/p_1, \ldots, m/p_n)$. One can then obtain each component e_i from $e_i(m/p_i) \equiv a \pmod{p_i}$. The congruences $a \pmod{p_i}$ for $i \in [n]$, may all be obtained via simultaneous modular reduction, i.e., the Chinese remainder algorithm reversed. Per Theorem 2.1.6, this cost of this recovery is softly linear in the size of the resulting univariate exponent, i.e., $\widetilde{\mathcal{O}}(\log(nD^n)) = \widetilde{\mathcal{O}}(n \log D)$. As in Kronecker substitution, the cost of this recovery is at most softly linear in the bit size of the resulting multivariate polynomial.

The caveat of this approach is that this approach has significant dependence on n. In Chapter 3, the univariate algorithms therein all had a runtime with cubic dependence on $\log D$, such that

this approach would have a $n^3 \log^3 D$ factor. If f is a black-box polynomial over $\mathbb{F}_q$, where $q > D$ and such that the multiplicative group $\mathbb{F}_q^*$ smooth order, then this may be reduced to $n^2 \log^2 D$ using *Prony's method* (Section 2.7).

We remark that Kronecker substitution is a dense interpolation technique, in that it will work regardless of the sparsity of f. In Chapter 5 we introduce techniques that may outperform Kronecker substitution when f is sufficiently sparse.

2.7 Prony's algorithm for interpolating black-box polynomials

In this section we redevelop the theory behind Prony's algorithm, originally due to Gaspard Clair François Marie Riche de Prony. Prony's algorithm has an interesting history. In 1795 in the first issue of the *Journal de l'École Polytechnique*, Prony gave an algorithm for the interpolation of a linear combination of sparse exponential functions. Independently, the development of decoding procedures for Bose–Chaudhury–Hocquenghem codes, beginning in the 1960s, eventually gave rise to sparse interpolation algorithms in the 1980s by Ben-Or–Tiwari [BT88] and others. Giesbrecht, Labahn, and Lee observed an equivalence between these methods [GLL02].

2.7.1 Linearly generated sequences

Definition 2.7.1. *A sequence* $\mathbf{a} = (a_0, a_1, a_2, \dots) \in \mathsf{R}^{\mathbb{N}}$*,* $i = 0, 1, 2, \dots$*, is said to be* **linearly generated** *over* K *if there exists values* $\psi_0, \dots, \psi_\ell \in \mathsf{K}$*, not all zero, such that for all* $i \geq 0$*,*

$$\sum_{j=0}^{\ell} a_{i+j}\psi_i = 0.$$

The polynomial $\Psi(y) = \sum_{j=0}^{\ell} \psi_j y^j$ *is called a* **linear generator** *of length* ℓ *for* a*. We say* Ψ **generates** $\boldsymbol{a}$ *and refer to* $\deg(\Psi) = \ell$ *as the* length *of the linear generator* Ψ*.*

The set of all linear generators of a sequence has the following useful structure:

Lemma 2.7.2. *Fix a sequence* $\mathbf{a} = (a_i)_{i \geq 0} \in \mathsf{R}^{\mathbb{N}}$ *and let* $\mathcal{I} \subset \mathsf{K}[y]$ *be the set of linear generators of* $\mathbf{a}$*. Then* $\mathcal{I}$ *forms an ideal in* $\mathsf{K}[y]$*.*

Proof. Let $\Psi = \sum_{j=0}^{\ell} \psi_j y^j$ and $\Phi = \sum_{j=0}^{m} \phi_m y^m$ be linear generators of the sequence $\mathbf{a}$. Without loss of generality, suppose $\ell \leq m$ and let $\psi_j = 0$ for $j > \ell$. Then for all $i \in \mathbb{Z}_{\geq 0}$, we have that

$$\sum_{j=0}^{m} a_{i+j}(\psi_j + \phi_j) = \sum_{j=0}^{\ell} a_{i+j}\psi_j + \sum_{j=0}^{m} a_{i+j}\phi_j.$$

As Φ and Ψ are linear generators for $\mathbf{a}$, the right-hand side of the equality above must be zero. It follows that $\Phi + \Psi$ is also a linear generator for $\mathbf{a}$.

Now suppose $g(y) = \sum_{j=0}^{n} g_j y^j \in \mathsf{K}[y]$. It remains to show that $g\Psi$ is a linear generator for $\mathbf{a}$. Write $g\Psi = h = \sum_{j=0}^{k+n} h_j y^j$ and observe, for $i \geq 0$, that

$$\sum_{j=0}^{\ell+n} a_{i+j} h_j = \sum_{j=0}^{n} g_j \left(\sum_{k=0}^{\ell} a_{i+j+k} \psi_k \right) = \sum_{k=0}^{n} g_j \cdot 0 = 0,$$

with the inner sums being zero as Ψ generates $\boldsymbol{a}$. □

If K is a field, then $\mathsf{K}[y]$ is a principal ideal domain, such that $\mathcal{I}$ is a principal ideal, i.e., $\mathcal{I}$ is an ideal generated by some polynomial Φ. $\mathcal{I}$ may be generated by any $\Phi \in \mathcal{I}$ of minimal degree. As Φ is a field we may take Φ to be monic.

Definition 2.7.3. *The* **minimal generator** *of a linearly generated sequence* $\mathbf{a}$ *is the monic linear generator* Φ *of* $\boldsymbol{a}$ *of minimal degree.*

The key observation that gives Prony's algorithm is that a t-sparse polynomial f, evaluated over an appropriate geometric sequence, gives a linearly generated sequence whose minimal generator Φ encodes the exponents of f.

Lemma 2.7.4. *Let* $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{K}[x_1, \ldots, x_n]$*, and* $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_n), \boldsymbol{\omega} = (\omega_1, \ldots, \omega_n) \in (\mathsf{K}^*)^n$*, where the values* $\boldsymbol{\omega}^{\boldsymbol{e}_i}$ *are distinct for* $i \in [t]$*. Then the sequence* $\boldsymbol{a}$ *given by* $a_i = f(\boldsymbol{\zeta}\boldsymbol{\omega}) = f(\zeta_1\omega_1, \ldots, \zeta_n\omega_n)$*,* $i \in \mathbb{Z}_{\geq 0}$ *is linearly generated with minimal generator*

$$\Phi(y) \stackrel{\text{def}}{=} \prod_{i=1}^{t} (y - \boldsymbol{\omega}^{\boldsymbol{e}_i}) = \prod_{i=1}^{t} (y - \textstyle\prod_{j=1}^{n} \omega_j^{e_{ij}}). \tag{2.9}$$

Proof. We prove by induction on t. If $t = 0$ then $f = 0$, which gives the sequence defined by $a_i = 0$, which is generated by 1. If $t = 1$, then $f = c\boldsymbol{x}^{\boldsymbol{e}}$, where $c \neq 0$. In which case the sequence given by $a_i = c(\prod_{j=1}^{n} \zeta_j)(\prod_{j=1}^{n} \omega_j^{e_j})^i$ is generated by $\Phi = \prod_{j=1}^{n} (y - \omega_j^{e_j})$. Φ is minimal in this case, as a generator of length 0 would imply that the a_i are all zero.

Now suppose the lemma holds for any t-sparse polynomial, and consider the $(t+1)$-sparse $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} + c_{t+1} \boldsymbol{x}^{\boldsymbol{e}_{t+1}}$. Then $\boldsymbol{a} = \boldsymbol{a}' + \boldsymbol{a}'' \subseteq \mathsf{K}^{\mathbb{N}}$, where

$$a'_i = \sum_{j=1}^{t} c_j \boldsymbol{\zeta}^{\boldsymbol{e}_j} \boldsymbol{\omega}^{\boldsymbol{e}_j}, \quad a''_i = c_{t+1} \boldsymbol{\omega}^{\boldsymbol{e}_{t+1}}.$$

By the induction hypothesis, $\mathbf{a}'$ and $\mathbf{a}''$ have minimal generators

$$\Phi' = \prod_{j=1}^{t}(y - \boldsymbol{\omega}^{\boldsymbol{e}_j}), \quad \Phi'' = y - \boldsymbol{\omega}^{\boldsymbol{e}_{t+1}}.$$

As $\Phi = \Phi'\Phi'' = \prod_{j=1}^{t+1}(y - \boldsymbol{\omega}^{\boldsymbol{e}_j})$ is a linear generator for both $\mathbf{a}'$ and $\mathbf{a}''$, it follows that Φ is a linear generator of $\mathbf{a}$.

Let $\Phi_{\min}$ be the minimal generator of $\mathbf{a}$. It remains to show that $\Phi = \Phi_{\min}$. Note that $\Phi_{\min}$ must divide Φ. Towards a contradiction, suppose that $(y - \boldsymbol{\omega}^{\boldsymbol{e}_{t+1}})$ is not a factor of $\Phi_{\min}$. Then $\Phi_{\min}$ divides Φ', and Φ' is a linear generator for $\mathbf{a}$ and hence $\mathbf{a}'' = \mathbf{a} - \mathbf{a}'$ as well. Observe that, by hypothesis, the linear factors $(y - \boldsymbol{\omega}^{\boldsymbol{e}_j})$, $j = 1, 2, \ldots, t+1$ are unique. Thus Φ'' is not a factor of Φ'. In other words, Φ' is not a linear generator of $\mathbf{a}''$, giving a contradiction. Thus $(y - \boldsymbol{\omega}^{\boldsymbol{e}_{t+1}})$ is a factor of $\Phi_{\min}$. As the choice of the $(t+1)$-th term was arbitrary, we have that $(y - \boldsymbol{\omega}^{\boldsymbol{e}_j})$ is a factor of $\Phi_{\min}$ for all j, and $\Phi = \Phi_{\min}$ as desired. □

Generally we will take $\boldsymbol{\zeta} = (1, 1, \ldots, 1)$; however, in some cases it will be advantageous to choose $\boldsymbol{\zeta}$ at random.

With or without the requirement in Lemma 2.7.4 that $\boldsymbol{\omega}^{\boldsymbol{e}} \neq 1$ for any $\boldsymbol{e} \in \mathbb{Z}^n_{\geq 0}$ with $e_i \leq \deg_{x_i}(f)$, the minimal generator of $\mathbf{a}$ would be the squarefree factor of maximal degree of $\prod_{j=1}^{n}(y - \omega_j^{e_j})$.

2.7.1.1 A Prony sparse polynomial identity test

We further remark that Lemma 2.7.4 gives a polynomial identity test for black-box polynomials with known bounds on their degree and sparsity. Namely, for a black-box polynomial $f \in \mathsf{Z}[x]_D$, where f is T-sparse and Z contains an element ω multiplicative order exceeding D, we can evaluate f over the geometric progression $\omega^i, i \in [0..T)$. As Z is a subset of a field (namely its *field of fractions*) and f has a linear generator of length T, the evaluations will all be zero if and only if f is zero. This test can generalize to the multivariate case via Kronecker substitution. Algorithm 2 gives this approach. We cite the following cost:

Algorithm 2: Polynomial identity testing of a sparse black-box polynomial

Input: A black-box polynomial $f \in \mathsf{Z}[x_1, \ldots, x_n]_D$; $T \geq \#f$; $\omega \in \mathsf{Z}$ of order at least $(D+1)^n$

Output: Accepts if and only if $f = 0$

1 **if** $f(\omega^i) = 0$ *for* $i = 0, 1, \ldots, T-1$ **then** Accept;

2 **else** Reject;

Lemma 2.7.5. *Algorithm 2 tests whether a black-box polynomial $f \in \mathsf{Z}[x_1, \ldots, x_n]$ is zero. Its cost is T black box queries.*

2.7.2 Determining the minimal generator

A linear generator $\Phi = y^s + \sum_{i=0}^{s-1} \phi_i y^i$ of the sequence $\boldsymbol{a}$ corresponds to a solution to the $s \times s$ *Hankel system*

$$\begin{bmatrix} a_k & a_{k+1} & \cdots & a_{k+s-1} \\ a_{k+1} & a_{k+2} & \cdots & a_{k+s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k+s-1} & a_{k+s} & \cdots & a_{k+2s-2} \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{s-1} \end{bmatrix} = - \begin{bmatrix} a_{k+s} \\ a_{k+s+1} \\ \vdots \\ a_{k+2s-1} \end{bmatrix}, \tag{2.10}$$

for any $k \in \mathbb{N}$. Usually we will choose $k = 0$ in (2.10). If $\mathbf{a}$ has a minimal generator of length $t \leq s$, then system (2.10) will have rank t.

One question we have not addressed yet is how to determine $t = \#f$ efficiently. When a bound $T \geq t$ is known, and our base ring and evaluations are exact (i.e., non-numerical) quantities, then the problem is solved via the **Berlekamp–Massey algorithm**.

The Berlekamp–Massey algorithm may be described in terms of the extended Euclidean algorithm and **Padé rational approximation**. Let $A(x) = a_0 x^{2T-1} + a_1 x^{2T-1} + \cdots + a_{2T-1}$. The minimal generator Φ is the monic polynomial of degree at most T such that

$$A\Phi \bmod x^{2T} = \Psi, \quad \deg(\Psi) < T.$$

In other words, we want to find $\Phi, \Psi \in \mathsf{K}[x]$ such that $A \equiv \Psi/\Phi \pmod{x^{2T}}$. The quotient Ψ/Φ is a rational approximation to A, known as the Padé approximation of order $(T-1, T)$ of A. This may be computed via the extended Euclidean algorithm acting on A and x^{2T}. The extended Euclidean algorithm produces, up to constant factors, all $(c, 2T-1-c)$ Padé approximations a/b such that a and b are coprime. The Berlekamp–Massey algorithm merely halts once we reach $c < T$. The Half-GCD algorithm can expedite Berlekamp–Massey. In particular, the order $(T-1, T)$ Padé approximation may be obtained in $\widetilde{\mathcal{O}}(T)$ K-operations from the output of half-GCD algorithm acting on x^{2T} and A. This gives us the following:

Lemma 2.7.6. *Given a linearly generated sequence $\mathbf{a} \in \mathsf{K}^{\mathbb{Z}_{\geq 0}}$ with a bound T on the length of its minimal generator $\Phi \in \mathsf{K}[y]$, one can find Φ in $\widetilde{\mathcal{O}}(T)$ field operations on an algebraic RAM.*

In the case that $\mathbf{a}$ is a sequence over an integral domain Z but not a field, the linear generator may be found without working in the field of fractions of Z, via the *fraction-free Berlekamp-Massey algorithm* due to Kaltofen and Yugahz [KY13]. This is an analogue to fraction-free algorithms for pseudoremainders of polynomials over integral domains, e.g., polynomials over $\mathbb{Z}$.

2.7.3 Prony's algorithm

We describe Prony's algorithm for an n-variate polynomial f over a field K. Suppose we are given black-box access to f as well as bounds $D \geq \deg(f)$ and $T \geq \#f$, and some $\boldsymbol{\zeta}, \boldsymbol{\omega} \in (\mathsf{K}^*)^n$, such that the values $\boldsymbol{\omega}^{\boldsymbol{e}} = \omega_1^{e_1} \cdots \omega_n^{e_n}$ are distinct for $\boldsymbol{e} \in \mathbb{Z}_{D+1}^n$. We set

$$a_i = f(\boldsymbol{\zeta}\boldsymbol{\omega}^i) = f(\zeta_1\omega_1^i, \ldots, \zeta_n\omega_n^i) \quad \text{for} \quad i \in [0..2T),$$

and let t be the largest rank appearing for any system (2.10), for $s \in [T]$. We then solve the Hankel system (2.10) for $s = t$, which produces the minimal generator $\Phi = y^t + \sum_{i=0}^{t-1} \phi_i y^i$. We factor $\Phi = \prod_{i=1}^{t}(y - \boldsymbol{\omega}_i^{\boldsymbol{e}})$, and from the values $\boldsymbol{\omega}_i^{\boldsymbol{e}}$ we obtain the multivariate exponents $\boldsymbol{e}_i$. The corresponding coefficients c_i may be obtained as solutions to the linear system

$$\begin{bmatrix} \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{0\boldsymbol{e}_1} & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{0\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{0\boldsymbol{e}_t} \\ \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{1\boldsymbol{e}_1} & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{1\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{1\boldsymbol{e}_t} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{(t-1)\boldsymbol{e}_1} & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{(t-1)\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^\top\boldsymbol{\omega}^{(t-1)\boldsymbol{e}_t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix}. \tag{2.11}$$

The determinant of the Vandermonde matrix appearing in (2.11) is $\prod_{1 \leq i < j \leq n}(\boldsymbol{\zeta}^\top(\boldsymbol{\omega}^{\boldsymbol{e}_i} - \boldsymbol{\omega}^{\boldsymbol{e}_j})$, such that the system produces a unique solution provided $\boldsymbol{\omega}^{\boldsymbol{e}_i} \neq \boldsymbol{\omega}^{\boldsymbol{e}_j}$ for $i \neq j$. Algorithm 3 gives a high-level description of Prony's method. We state the following cost:

Lemma 2.7.7. *Algorithm 3 (Prony) interpolates a black-box polynomial $f \in \mathsf{Z}[x_1, \ldots, x_n]$ with a query-cost of T queries.*

We remark that Prony's formulation of Algorithm 3 was for sums of dampened exponentials. Namely, Prony's algorithm was to interpolate functions of the form $h : \mathbb{C} \to \mathbb{C}$ of the form

$$h = \sum_{j=1}^{t} c_j \exp{(d_j + e_j i)t},$$

where $c_i \in \mathbb{C}, d_i, e_i \in \mathbb{R}$, and $i = \sqrt{-1}$. Sparse polynomial interpolation is the specific case that the *damping factors* $d_j = 0$ and the values $e_j \in \mathbb{Z}$ for all $j \in [t]$. Prony's algorithm, as originally formulated by Prony, evaluates Ψ over a real arithmetic progression of length $2T$. Then evaluating Ψ over an arithmetic progression $t = u, u+v, u+2v, \ldots$, where $u, v \in \mathbb{R}$, is equivalent to evaluating the polynomial

$$h_1(x) = \sum_{j=1}^{t} c_j x^{e_j}$$

over the geometric progression $x = \exp(u + vj), j \in (1, 2, \ldots)$.

There are a number of variants of Prony's algorithm. We describe some of these variants in the remainder of Section 2.7.

Algorithm 3: Prony's algorithm for interpolating a T-sparse black-box polynomial

Input: A black-box polynomial $f \in \mathsf{K}[x_1, \ldots, x_n]$; $D \geq \max_i \deg_{\boldsymbol{x}_i} f$; $T \geq \# f$; $\boldsymbol{\zeta}, \boldsymbol{\omega} \in (\mathsf{K}^*)^n$, where the values $\{\boldsymbol{\omega}^{\boldsymbol{e}} : \boldsymbol{e} \in [0..D]^n\}$ are distinct.

Output: A sparse representation of f.

1 Choose $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n) \in \mathsf{K}^n$ with $\mathrm{ord}(\omega_i) > D$ for $i \in [n]$, and $k \in \mathbb{Z}_{\geq 0}$;
2 **for** $i \leftarrow 0$ **to** $2T - 1$ **do** $a_{i+k} \leftarrow f(\boldsymbol{\zeta}\boldsymbol{\omega}^{i+k})$;
3 Compute minimal generator $\Phi(y)$ of length $t \leq T$ to the sequence given by $(a_k, \ldots, a_{k+2T-1})$;
4 Factor Φ to obtain its roots $b_1, \ldots, b_t \in \mathsf{Z}$;
5 **for** $i \leftarrow 1$ **to** t **do** Determine exponent $\boldsymbol{e}_i$ from $b_i = \boldsymbol{\omega}^{\boldsymbol{e}_i}$;
6 Determine coefficients $(c_1, \ldots, c_t)$ via the Vandermonde system (2.11), if the system is nonsingular, otherwise return `fail`;
7 **return** $\sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i}$;

2.7.4 Decoding BCH codes and Prony's algorithm over finite fields

Decoding procedures for Reed–Solomon [RS60] and **Bose–Chaudhuri–Hocquenghem (BCH) codes** [BR60] were developed by Peterson, Berlekamp, Gorenstein (amongst others), and and subsequently unified under an algebraic treatment by Blahut [Bla83]. These decoding procedures give a means of interpolating a sparse black-box polynomial.

A BCH code is an additive subgroup of a vector space over a finite field $\mathbb{F}_q$. We can construct a BCH code with distance ℓ as follows. We select an element $\alpha \in \mathbb{F}_{q^s}$ belonging to a field extension of $\mathbb{F}_q$. For some choice of $k, \ell \in \mathbb{Z}_{\geq 0}, \leq s = \mathrm{ord}(\alpha)$, we take the *generator* of the code to be

$$g(x) = \mathrm{lcm}\left(m_{\alpha^{k+1}}(x), m_{\alpha^{k+2}}(x), \ldots, m_{\alpha^{k+D-1}}(x)\right) \in \mathbb{F}_q[x],$$

where $m_\beta \in \mathbb{F}_q[x]$ is the minimal polynomial of $\beta \in \mathbb{F}_{q^s}$ over $\mathbb{F}_q$. The code is the additive subgroup of $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ generated by g, where n is the multiplicative order of α. We identify $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ with $\mathbb{F}_q^n$ as additive groups. A Reed–Solomon code is merely a BCH code where α is selected from $\mathbb{F}_q$.

Every *code word* $\mathcal{C} \in \mathbb{F}_q[x]$ has the property that $\mathcal{C}(\alpha^i) = 0$ for $i \in [k..k + D)$. Thus, for a code word and error term $\mathcal{C}, \mathcal{E} \in \mathbb{F}_q[x]_{2t}$, the evaluation $(\mathcal{C} + \mathcal{E})(\alpha^i)$ gives us evaluation $\mathcal{E}(\alpha^i)$, for $i \in [k..k + D)$. Moreover, for any pair of code words $\mathcal{C} \neq \mathcal{C}'$, we have that $(\mathcal{C} - \mathcal{C}')(\alpha^j)$ must be nonzero for some $j \in [d..n]$, such that $(\mathcal{C} - \mathcal{C}')$ has a linear generator of length at least D, and that $(\mathcal{C} - \mathcal{C}')$ is D-sparse, and the code has distance D.

If we select $d = 2t + 1$, then we can decode a code word with at most t errors, by essentially interpolating the t-sparse error $\mathcal{E} \in \mathbb{F}_q[x]$ of degree less than n from its evaluations $\alpha^k, \ldots, \alpha^{2t+k}$.

In the setting of BCH code error-correction, the auxillary polynomial is known as the **error-locator polynomial**, as it enables us to find the locations (exponents) of the error $\mathcal{E}$.

More generally, Prony's method allows for the sparse interpolation of arbitrary black-box sparse polynomials over finite fields. In the case that the interpolant $f \in \mathbb{F}_q[x]$ is univariate, by choosing ω_1 to have multiplicative order exceeding D, we guarantee that the transposed Vandermonde system (2.11), such that can deterministically interpolate f.

For the interpolation of an n-variate function f use Kronecker substitution; however, in order to guarantee a nonsingular linear system (2.11), we would require a root of unity of multipicative order at least $(D+1)^n$. That is, we would need to evaluate f in a field extension containing more than $(D+1)^n$ elements. We can also probabilistically guarantee nonsingularity over $\mathbb{F}_q$, independently of n, by the following proposition due to Zippel.

Proposition 2.7.8 (Prop 8, [Zip90]). *Let $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_t \in \mathbb{N}^n$ be exponents with total degree less than D^*, for $i \in [t], j \in [n]$. For $\boldsymbol{\zeta} \in (\mathbb{F}_q^*)^n$, and $\boldsymbol{\omega} \in \{\mathbb{F}_q\}^n$ chosen uniformly at random, the linear system (2.11) is nonsingular with probability at least $1 - \frac{1}{2}t(t-1)D^*/q$.*

The bottleneck of Prony's algorithm over finite fields is the recovery of the exponents $\boldsymbol{e}_i$ from the roots $\omega^{\boldsymbol{e}_i}$ of the auxillary polynomial. Discrete logarithms with over arbitrary finite fields are not known to be solveable in polynomial time in general on a classical computer.

Discrete logarithms can be computed efficiently, however, in the case that the base $\omega \in \mathbb{F}_q$ has **smooth** multiplicative order d, throughout meaning that all prime factors of d are bounded by a constant. We say an integer $d \in \mathbb{Z}_{>0}$ is B**-smooth** if all of its prime factors are at most B. If d is smooth, then a discrete logarithm with base ω can be performed in $\mathcal{O}(\log q)$ $\mathbb{F}_q$-operations, or $\mathcal{O}(\log^2 q)$ bit operations. Kaltofen observed in [Kal10] that this approach could expedite sparse interpolation of a black-box polynomial $f \in \mathbb{Z}[\boldsymbol{x}]$, by embedding f in an appropriate field $\mathbb{Z}_q$, where q is prime. The task of finding a prime q for which $q-1$ contains a smooth factor exceeding D, however, is nontrivial and can potentially dominate the cost of interpolation.

2.7.5 The Ben-Or–Tiwari Algorithm

In [BT88], the authors give a deterministic algorithm for the interpolation of a sparse black-box polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$, with bounds $T \geq t = \#f$ and $D \geq \max_{i=1}^n \deg_{x_i}(f)$. Perhaps the most significant contribution of [BT88] was extending Prony's algorithm to multivariate polynomials.

In this setting, for all $i \in [n]$, we set $\zeta_i = 1$ and ω_i is chosen to be the ith prime which we will denote here by p_i. The exponents $\boldsymbol{e}_i$ are then recovered via the prime factorization of the roots $\boldsymbol{\omega}^{\boldsymbol{e}_i}$ of the auxillary polynomial Φ. By the choice of $\boldsymbol{\omega}$, we have that $\boldsymbol{\omega}^{\boldsymbol{e}_i} \neq \boldsymbol{\omega}^{\boldsymbol{e}_j}$ for distinct exponents $\boldsymbol{e}_i \neq \boldsymbol{e}_j$, such that the resulting transposed Vandermonde system (2.11) is nonsingular.

In [KY89], Kaltofen and Lakshman made improvements to runtime of the Ben-Or–Tiwari algorithm based on faster transposed-Vandermonde system solving and root-finding subroutines.

2.8 Early termination, sparsity testing, and sparsity estimation

In [KL03], Kaltofen and Lee studied black-box sparse interpolation in the case that no bound $T \geq t = \#f$ is given. They developed the technique of **early termination**, which probabilistically detects when we have enough black-box queries to interpolate f. One can probabilistically test that f is not 0-sparse by way of the DLSZ identity test. To test whether f is s-sparse, $s > 1$ consider the symbolic sequence $\boldsymbol{a} \in \mathbb{F}_q[\boldsymbol{x}]^{\mathbb{N}}$ given by

$$a_i = f(x_1^i, \ldots, x_n^i) \in \mathbb{F}_q[x_1, \ldots, x_n].$$

We can think of a_i as an evaluation of f at the symbolic point $(x_1^i, \ldots, x_n^i)$. It follows that the a_i satisfies the Hankel relationship (2.10) for $s \geq t$. For $s > 0$, define the Hankel matrices of polynomials

$$\mathbf{H}_s \stackrel{\text{def}}{=} \left[a_{i+j}\right]_{i,j=0}^{s-1}.$$

If $s > t$, then we have that $\mathbf{H}_s$ is singular. In other words, $\det(\mathbf{H}_s) \in \mathsf{R}[\boldsymbol{x}]$ is identically zero, such that $\det(\mathbf{H}_s)(\boldsymbol{\omega}) = 0$ for any $\boldsymbol{\omega}$ with components taken from the algebraic completion of $\mathbb{F}_q$.

If $s \in [1..t]$, then we can show that $\mathbf{H}_s$ is probably full rank. Write the determinant of $\mathbf{H}_s$ as

$$\det(\mathbf{H}_s) = \sum_{\sigma} (-1)^{\operatorname{sign}(\sigma)} \prod_{i=1}^{s} \left(\sum_{j=1}^{t} c_j \boldsymbol{x}^{\sigma(i)\boldsymbol{e}_j} \right) \in \mathbb{F}_q[\boldsymbol{x}], \tag{2.12}$$

where the sum is taken over all permutations σ acting on $[s]$. Suppose $\boldsymbol{e}_1 \succ \boldsymbol{e}_2 \succ \cdots \succ \boldsymbol{e}_t$ with respect to the lexicographical ordering. Then the expansion of (2.12) contains a unique term of degree $\boldsymbol{d} = \boldsymbol{e}_2 + (2 + 3 + \cdots + s)\boldsymbol{e}_1$, namely $c_1^{(s-1)} c_2 \boldsymbol{x}^{\boldsymbol{d}}$.

We may bound the degree of $\det(\mathbf{H}_s)$ by $s(s-1)D^*$, such that, for $\boldsymbol{\omega}$ chosen uniformly at random from $\mathbb{F}_q^n$, $\det(\mathbf{H}_s)(\boldsymbol{\omega})$ is nonzero with probability at least $s(s-1)D^*/q$ by the DLSZ Lemma. This sparsity test (Algorithm 4) was later independently discovered in [GJR10], who were studying it in the specific context of property testing.

Lemma 2.8.1. *Algorithm 4 accepts if $f \in \mathbb{F}_q[\boldsymbol{x}]$ is s-sparse and rejects with probability at least $2/3$ otherwise. Its cost is $\mathcal{O}(s)$ black-box queries to f and an additional $\widetilde{\mathcal{O}}(s \log q) \supseteq \widetilde{\mathcal{O}}(s \log D^*)$ bit operations.*

Algorithm 4: A probabilistic sparsity test

Input: $f \in \mathbb{F}_q[\boldsymbol{x}]$, $D^* \geq \deg(f)$ and $s \in \mathbb{Z}_{>0}$, where $q \geq 3s(s-1)D$.
Output: If f is s-sparse, then the algorithm accepts. If f is not s-sparse, then the algorithm rejects with probability at least $2/3$.

1 Choose $\boldsymbol{\omega} \in \mathbb{F}_q$ uniformly at random;
2 **for** $i \in [0..2s-2]$ **do** $a_i \leftarrow f(\boldsymbol{\omega}^i)$;
3 $\mathbf{H}_s \leftarrow [a_{i+j}]_{i,j=0}^{s-1}$;
4 **if** $\mathbf{H}_s$ *is non-singular* **then** Accept;
5 **else** Reject;

Proof. Probabilistic correctness follows from the preceding discussion. The query complexity is immediate. The cost of computing the Hankel determinant is $\widetilde{\mathcal{O}}(s)$ operations in $\mathbb{F}_q$, or $\widetilde{\mathcal{O}}(s \log q) \in \widetilde{\Omega}(s \log D^*)$ bit operations. □

It follows that, if we test whether $\det(\mathbf{H}_s) = 0$ for $s = 1, 2, 3, \ldots, t+1$, the probability that $\det(\mathbf{H}_{t+1})$ is not the first nonsingular Hankel matrix is at most

$$\frac{D^*}{q} \sum_{s=1}^{t} s(s-1) = \frac{Dt(t-1)(t+1)}{3q}.$$

Thus, by choosing a random evaluation point $\boldsymbol{\omega} \in \mathbb{F}_q^n$, one may probabilistically compute t as the least value such that $\mathbf{H}_{t+1}$ is nonsingular.

In some instances, we may only want an estimate for t that is tight up to a constant factor. In this case one can employ a technique we call **repeated doubling**. We make an initial guess $t = T_{\min}$, and run Algorithm 4 (alternatively, a sparsity test of one's choosing) for $s = T_{\min}, 2T_{\min}, 4T_{\min}, \ldots$, until the algorithm rejects. In this case, the probability that the first instance of rejection is for $s \in (t..2t]$ is

$$\frac{D^*}{q} \sum_{i=0}^{\lfloor \log t \rfloor} 2^i(2^i - 1) < \frac{D^* t^2}{q}. \tag{2.13}$$

Algorithm 5 gives such a method for the estimation of the sparsity $t = \#f$ of a black-box polynomial $f \in \mathbb{F}_q[x]$. Its correctness and cost are as follows.

Lemma 2.8.2. *Algorithm 5 behaves as stated. Its cost is $\widetilde{\mathcal{O}}(t + T_{\min})$ black-box queries and an additional $\widetilde{\mathcal{O}}(t \log q)$ bit operations.*

We further remark that Algorithm 5 can be made to succeed with probability $1 - \mu$, at the cost of an additional $\mathcal{O}(\log(\mu^{-1}))$ factor as usual; we merely take the maximum output after running the algorithm $\mathcal{O}(\log(\mu^{-1}))$ times.

Algorithm 5: Black-box sparsity estimation via repeated doubling

Input: $f \neq 0 \in \mathbb{F}_q[\boldsymbol{x}]$, a black-box polynomial; $T \geq t = \#f$ and $D^* \geq \deg(f)$, where $q \geq 3D^*T^2$; $T_{\min}$, an initial guess for t.

Output: If $T_{\min} > t$, algorithm produces $T_{\min}$. If $T_{\min} \leq t$, then with probability at least $2/3$, algorithm produces $s \in (t..2t]$, and otherwise produces $s \leq t$

Choose $\boldsymbol{\omega} \in \mathbb{F}_q^n$ independently and uniformly at random;
$s \leftarrow T_{\min}$;
for $i \leftarrow 0$ **to** $2s-2$ **do** $a_i \leftarrow f(\boldsymbol{\omega}^i)$;
while `true` **do**
    for $i \leftarrow s-1$ **to** $2s-2$ **do** $a_i \leftarrow f(\boldsymbol{\omega}^i)$;
    $\boldsymbol{H} \leftarrow [a_{i+j}]_{i,j=0}^{\widetilde{t}-1}$;
    if $\boldsymbol{H}$ *is singular* **then return** s;
    else $s \leftarrow 2s$;

Few things are harder to put up with
than a good example.

Mark Twain

Chapter 3

Sparse interpolation of straight-line programs

3.1 Introduction

A *straight-line program (SLP)* is a branchless sequence of arithmetic instructions that can model a rational function. Straight-line programs are central to the study of algebraic complexity. One reasonable measure of the complexity of a rational function is the least size of all straight-line programs that compute it.

This chapter discusses the sparse interpolation of a SLP: learning the sparse representation of a polynomial f given by a SLP. In 2009, Garg and Schost gave an algorithm that solves this problem, given as inputs asymptotically tight bounds on the degree and sparsity of f, with deterministic running time polynomial with respect to the combined size of the inputs and outputs [GS09]. In subsequent work [GR11], Giesbrecht and Roche developed a new technique, *diversification*, which enabled them to give a faster, Las Vegas method.

In this chapter we will describe two Monte Carlo sparse interpolation algorithms for SLPs. The methods presented herein have the advantage over previous variants in that they are all softly linear in the number of terms in the output. This work was published in a series of papers [AGR13; AGR14] with Mark Giesbrecht and Daniel S. Roche.

SLPs can serve as a generalization of black-box polynomials, producing arbitrary homomorphic images of f, not limited to the evaluation homomorphism. A natural generalization of polynomial interpolation is the recovery of f given by specified sets of homomorphic images. The algorithms herein all recover f, essentially by Chinese remaindering on a set of modular images that each preserve the sparseness of f. In order to construct these modular images for f over

an integral domain Z, it suffices that we can evaluate f over the *algebraic completion* of Z. We describe this model in terms of extended black-box polynomials, for which we easily generalize the algorithms of [AGR13; AGR14].

These techniques may expedite sparse polynomial arithmetic and any R-algebraic program whose output is a sparse polynomial, or one that will generate sparse polynomials as results to intermediate computation. We will use many of the techniques developed in this chapter to create fast algorithms for sumset and sparse polynomial multiplication, described in Chapter 4.

We further believe that techniques developed herein may have applications in signal processing and multi-exponential analysis. A continuous-time signal comprised of a sparse linear combination of integer-valued frequencies can be expressed as a complex-valued black-box polynomial f with evaluation access over the unit circle. We show in the conclusions how to adapt our algorithms to this model to recover such f on an algebraic RAM.

3.2 Preliminaries

In this section we discuss the extended black box polynomial model and how our algorithms will interact with it. For the purposes of this chapter we will restrict our attention to univariate extended black-box polynomials.

We recall, from Definition 1.4.2, that a univariate extended black-box polynomial for $f \in \mathsf{R}[x]$ takes as input $a \in \mathsf{R}[y_1, \ldots, y_m]$ and $b_1 \in \mathsf{R}[y_1], \ldots, b_m \in \mathsf{R}[y_m]$ and produces $f(a) \bmod \mathcal{B}$, where $\mathcal{B} = \langle b_1, \ldots, b_m \rangle \subset \mathsf{R}[y_1, \ldots, y_m]$ is the ideal generated by the b_i.

An extended black box may be simulated by an SLP. Given a straight-line program computing $f \in \mathsf{R}[x]$ and $\alpha \in \mathsf{R}$, we may compute $f(\alpha x) \bmod \mathcal{B}$ by replacing any instance of x appearing as an input to an instruction with αx, and performing each resulting arithmetic instruction over $\mathsf{R}[x]/\mathcal{B}$.

3.2.1 Extended black box queries

Here we are strictly interested in images of the form $f(ax) \bmod (x^p - 1)$, for constants $a \in \mathsf{R}$ and $p \in \mathbb{Z}_{>0}$. Note, for $f = \sum_{i=1}^{t} c_i x^{e_i} \in \mathsf{R}[x]$, that in its reduced form,

$$f \bmod (x^p - 1) = \sum_{i=1}^{t} c_i x^{e_i \bmod p}. \tag{3.1}$$

As such we introduce the shorthand notation,

$$f^{\bmod p} \stackrel{\text{def}}{=} f(x) \bmod (x^p - 1).$$

We may think of $f^{\bmod p}$ as being either in $\mathsf{R}[x]$ or $\mathsf{R}[x]/\langle x^p - 1\rangle$ and will specify which is the case if relevant.

Definition 3.2.1. *Given an extended black-box polynomial $f \in \mathsf{R}[x]$, we let*

$$f^{\bmod p} \stackrel{\text{def}}{=} f \bmod (x^p - 1).$$

We will call an image $f^{\bmod p}$ a **query** *of degree p or simply a p-***query***, and we call an image $f(\alpha x)^{\bmod p}$ a* **shifted query** *of degree p or merely a* **shifted** *p-***query***). We call α the* **shift** *of the query, e.g., a p-query is a shifted query with shift 1. We may write $\mathcal{O}(\psi(n))$-***query** *to denote a p-query for some $p \in \mathcal{O}(\psi(n))$ and similarly for all other query variants defined herein and all asymptotic classes of functions over $\mathbb{Z}_{>0}$.*

An extended black-box for f can produce (shifted) queries of arbitrary degree. We will often but not always choose the query degree p to be prime from a specified range $(\lambda..2\lambda]$. While our algorithms were originally designed for straight-line programs, it is straightforward to generalize these algorithms to extended black-box polynomials. Much of our analysis will be towards selecting an appropriate λ such that the probability $f^{\bmod p}$ satisfies certain properties for a randomly selected prime from $(\lambda..2\lambda]$.

In order to produce a shifted query $f(\alpha x)^{\bmod p}$, given an SLP that computes f, we evaluate $\mathbf{\Gamma}$ at $\alpha x \in \mathsf{R}[x]/\langle x^p - 1\rangle$. To compute this evaluation, each arithmetic instruction of the SLP is performed in $\mathsf{R}[x]$ and then reduced via division by $(x^p - 1)$, all using dense arithmetic. By Theorem 2.1.2, each of these instructions cost $\widetilde{\mathcal{O}}(p)$ R-operations. The resulting dense image can then be converted to a sparse representation in $\widetilde{\mathcal{O}}(p)$ R and bit operations.

Lemma 3.2.2. *Let $\mathbf{\Gamma}^f$ be a straight-line program of length L that computes f, $\alpha \in \mathsf{R}$, and $p \in \mathbb{Z}_{>0}$. Then, given $\mathbf{\Gamma}^f$, we can compute a (shifted) p-query $f(\alpha x)^{\bmod p}$ using $\widetilde{\mathcal{O}}(Lp)$ ring operations.*

We can also produce shifted queries from the sparse representation of f itself. We first write $f(\alpha x)$ in place of $f(x)$ by replacing each term $cx^e \in f$ with $c\alpha^e x^e$. This costs $\widetilde{\mathcal{O}}(T \log D)$ bit operations by a square-and-multiply approach. We then merely reduce the exponents of $f(\alpha x)$ modulo p, and then sort the resulting terms and add terms of equal degree together.

Lemma 3.2.3. *Given a sparse representation of $f \in \mathsf{R}[x]$, we can produce a p-query of f with cost $\widetilde{\mathcal{O}}(T \log D)$ bit operations and an additional $\widetilde{\mathcal{O}}(T)$ ring operations; and a shifted p-query with a cost of $\widetilde{\mathcal{O}}(T \log D)$ bit and ring operations.*

Remark 3.2.4. *We will generally suppose that a (shifted) p-query of f has cost at least that of querying f via its sparse representation. By this assumption and the previous lemma, a p-query costs at least $\widetilde{\Omega}(T \log D)$ bit operations and $\widetilde{\Omega}(T)$ R-operations, whereas a shifted p-query costs at least $\widetilde{\Omega}(T \log D)$ bit and ring operations. Note this cost exceeds the space complexity of $f^{\bmod p}$ ($f(\alpha x)^{\bmod p}$). We will measure the cost of the algorithms with respect to the number of queries and shifted queries. We also will keep track of additional ring and bit operations, however, under this assumption the query cost will dominate all other costs in all algorithms described this chapter.*

3.2.2 Hashing exponents

Note that by (3.1), reduction modulo $(x^p - 1)$ effectively acts as a hash function on the exponents of f, mapping a term of degree e to one of degree $e \bmod p$. We generalize the maps to a broader family of *sparsity-preserving maps*, which we introduce here in order to describe the technique of diversification. We revisit these maps in Chapter 5.

Definition 3.2.5. *We say a map $\Phi : \mathsf{R}[x] \to \mathsf{R}[x]$ is a* **sparsity-preserving map** *if it is a* R*-module homomorphism for which there exists $\Phi^* : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $\Phi(x^d) = x^{\Phi^*(d)}$. We call Φ^* the* **induced map** *of Φ.*

For $f = \sum_{i=1}^{t} c_i x^{e_i} \in \mathsf{R}[x]$, we let the **preimage** *of a* formal term *cx^d of $\Phi(f)$ (i.e. such that c may be 0) be the sum of all nonzero terms of f which map to terms of degree d, i.e.,*

$$\sigma = \sum_{\substack{i \in [t] \\ \Phi^*(e_i) = d}} c_i x^{e_i}.$$

In the case Φ is the R-linear map given by $x^e \mapsto x^{e \bmod p}$, the induced map of Φ is the map $\Phi^* : e \mapsto e \bmod p$. We will construct images of f under sparsity-preserving maps, and learn the exponents of the terms of f from information given by the induced map acting on the exponents. Via this induced map, Φ acts as a hash function on the exponents of f, mapping them from $[0..D]$ to the appreciably smaller set $[0..p)$.

Definition 3.2.6. *Let $\Phi : \mathsf{R}[x] \to \mathsf{R}[x]$ be a sparsity-preserving map with induced map Φ^*, $f \in \mathsf{R}[x]$, and τ_1 a term of f.*

We say τ_1 is a **singleton** *(with respect to Φ) if there does not exist another nonzero term $\tau_2 \in f$ such that $\Phi(\tau_1)$ and $\Phi(\tau_2)$ are terms of the same degree. If there does exist such a term τ_2, we say that τ_1 and τ_2* **collide***, and the sum of terms $\tau \in f$ such that $\deg(\Phi(\tau)) = deg(\Phi(\tau_1))$ form a* **collision***. Moreover, we say a term $\tau \in \Phi(f)$ is a* singleton *if a singleton of f uniquely maps to it.*

We say an exponent $e \in \mathrm{supp}(f)$ is a singleton *or* collides *if its corresponding term does. In the case that Φ is given by $f \mapsto f^{\bmod p}$ and Φ^* by $e \mapsto e \mod p$, we will say that $\tau_1 = cx^e \in f$ is a singleton or collides* modulo $(x^p - 1)$ *and its exponent is a singleton or collides* $\bmod p$.

Example 3.2.7. *Define $\Phi : \mathsf{R}[x] \to \mathsf{R}[x]$ by $x^e \mapsto x^{e \bmod 5}$. Consider $f = 3x^2 + 4x^3 + 5x^8 - 3x^{17} + 7x^{19}$. Then $\Phi(f) = f^{\bmod 5} = 9x^3 + 7x^4$. We have that $3x^2$ collides with $-3x^{17}$ to give the formal term $0x^2$, and that $4x^3$ collides with $5x^8$ to give $9x^3$. The remaining term of f, $7x^{19}$, is a singleton mapping to $7x^4$.*

The preimage of $9x^3$ is the sum of terms of f with exponent congruent to $3 \pmod 5$, i.e. $\sigma = 4x^3 + 5x^8$. The preimage of the formal term $0x^2$ of $f^{\bmod 5}$ is $\sigma = 3x^2 - 3x^{17}$.

We stress, per the example, that terms with zero coefficients appearing in images may have nonzero preimages.

Algorithm	Reference[1]	Section	Ring[2]	Queries[3]	Query deg.[3]	SLP Cost[4]	Type[5]
Dense			R	1	D	LD	Det
Alg. 6	[GS09]	3.3.1	R	$T^2 \log D$	$T^2 \log D$	$LT^4 \log^2 D$	Det
Alg. 7		3.3.2	R	$T \log D$	$T^2 \log D$	$LT^3 \log^2 D$	LV
Alg. 9	[GR11]	3.3.4	$\mathbb{F}_q$	$\log D$	$T^2 \log D$	$LT^2 \log^2 D$	LV
IterativeSI	[AGR13]	3.4	R	$\log D \log T$	$T \log^2 D$	$LT \log^3 D$	MC
Alg. 10	[AGR14]	3.5	$\mathbb{F}_q$	$\log D \log T$	$T \log D$	$LT \log^2 D$	MC
SparsePIT	[Bl+09]	3.3.3	R	$T \log D$	$T \log D$	$LT^2 \log^2 D$	Det

1 GS09 (Garg–Schost); GR11 (Giesbrecht–Roche); AGR13/14 (Arnold–Giesbrecht–Roche); Bla+09 (Bläser–Hardt–Lipton–Vishnoi)
2 R = any commutative ring with identity.
3 # of queries and query-degree in terms of "$\widetilde{\mathcal{O}}$". Over Z queries are shifted queries.
4 Cost on a length-L SLP in terms of soft-oh number of ring operations.
5 Det = deterministic; LV = Las Vegas; MC = Monte Carlo.

Table 3.1: Algorithms for the sparse interpolation and identity testing of an extended black box polynomial

Algorithm	Soft-oh cost
Dense	$LD \log q$
Alg. 6	$LT^4 \log^2 D \log q$
Alg. 7	$LT^3 \log^2 D \log q$
Alg. 9	$LT^2 \log^2 D(\log D + \log q)$
Alg. **IterativeSI**	$LT \log^3 D \log q$
Alg. 10	$LT \log^2 D(\log D + \log q)$
SparsePIT	$LT^2 \log^2 D \log q$

Table 3.2: Bit cost of sparse interpolation and identity testing algorithms for length-L SLPs over $\mathbb{F}_q$

3.2.3 Organization of chapter and summary of results

Table 3.1 gives a summary of results for sparse interpolation, sparsity testing, and sparse identity testing of extended black-box polynomials, as well as giving relevant references and the sections where each algorithm appears in this chapter. Some of the algorithms work for f over arbitrary commutative rings with identity R; other algorithms only work for polynomials over a finite field $\mathbb{F}_q$.

The algorithms over $\mathbb{F}_q$ rely on diversification, a technique whose correctness relies on the DLSZ Lemma. As such, one can generalize these algorithms for any coefficient ring R containing

a *regular-difference* set of cardinality $(1+\epsilon)v$ for some $\epsilon > 0$. In order to compare these algorithms in a more even setting, Table 3.2 compares the soft-oh bit cost of each of the algorithms acting on an SLP over $\mathbb{F}_q$. From Table 3.2 we see that diversification essentially replaces a cost factor of $\log D \log q$ with $(\log D + \log q)$.

In Section 3.3, we describe previous work on the interpolation and identity testing of straight-line programs. In Section 3.3.1 we give the algorithm of Garg and Schost, which achieved a ring-operation cost that was polynomial in the input and output size. In 3.3.2 we describe a Monte Carlo variant thereof that saves a $\log D$ cost factor. This method can be made Las Vegas via polynomial identity testing (PIT). We give a deterministic PIT method due to Bläser et al., which, given a length-L SLP $\mathbf{\Gamma}^f$ and bounds D and T, tests whether f is zero with cost $\widetilde{\mathcal{O}}(LT^2 \log^2 D)$. In Section 3.3.4, we describe the technique of diversification and give a Monte Carlo algorithm of Giesbrecht and Roche. This algorithm may also be made Las Vegas via the test of Bläser et al.

In Section 3.4, we describe a new Monte Carlo approach due to the author, Giesbrecht, and Roche, that achieves runtime that is softly linear in the sparsity T. In Section 3.5, we give a faster algorithm in the case that the extended black-box is over a integral domain of sufficient order. These algorithms are Monte Carlo as their costs are dominated by that of polynomial identity testing (PIT). Our algorithms require as inputs bounds on the degree and sparsity of f. We give algorithm to determine nontrivial bounds on the partial degrees and sparsity of multivariate f given by an SLP in Section 3.6. In Section 3.7, we offer conclusions and open problems.

3.3 Previous Work

3.3.1 The Garg–Schost algorithm

In 2009, in the paper [GS09], Sanchit Garg and Éric Schost gave an algorithm that interpolates a sparse polynomial given by a straight-line program $\mathbf{\Gamma}^f$, given bounds on the sparsity and degree of f. Provided these bounds are tight up to polynomial factors, their algorithm is the first known algorithm that achieves a cost, in terms of ring operations, that is polynomial in the combined size of the input and output. We state the cost of the algorithm as a Theorem.

Theorem 3.3.1. *Algorithm 6 takes an extended black-box for $f \in \mathsf{R}[x]$ and bounds $D \geq \deg(f)$, and $T \geq \#f$, and constructs a sparse representation for f. Its cost is $\widetilde{\mathcal{O}}(T^2 \log D)$ queries of degree $\mathcal{O}(T^2 \log D)$.*

Definition 3.3.2. *We say a prime p is a* **good prime** *(for f) if none of the terms of f collide modulo $(x^p - 1)$. Otherwise we say p is a* **bad prime***. If p is a good (bad) prime we call an image $f^{\bmod p}$ a* **good image** *(***bad image***, resp.).*

Algorithm 6: The Garg–Schost method for extended black-box polynomials

Input: An extended black-box polynomial $f \in \mathsf{R}[x]$; bounds $T \geq \#f, D \geq \deg(f)$.
Output: A sparse representation of f.

$\ell \leftarrow \lceil \frac{1}{2}T(T-1)\log^2 D + T\log(D+1) + 1 \rceil$; // $\ell =$ bound on # of queries needed

// Construct images $f^{\bmod p}$ for a set of good primes $p \in \mathcal{P}$
$p_1, \ldots, p_\ell \leftarrow$ the least ℓ primes;
$t \leftarrow \max_{i=1}^{\ell}(\#f^{\bmod p_i})$;
$\mathcal{P} \leftarrow \{p_i : i \in [\ell], \#f^{\bmod p_i} = t\}$

// Construct images of $\Lambda \bmod p$ for good primes $p \in \mathcal{P}$
$M \leftarrow 1$; // $M =$ combined modulus of congruences in $\mathcal{C}$
for $p \in \mathcal{P}$ **do**
    $\Lambda_p \leftarrow \prod_{e \in \mathrm{supp}(f^{\bmod p})}(y - e) \bmod p$;
    $M \leftarrow Mp$;
    if $M > 2D^T$ **then break**;

// Construct exponents e_i and pair each e_i with its coefficient
Construct $\Lambda \in \mathbb{Z}[x]$ with height at most D^T from congruences $\{\Lambda_p \bmod p\}$;
Factor $\Lambda = \prod_{j=1}^{t}(y - e_j)$ to get exponents e_j;
for $j = 1, 2, \ldots, t$ **do** $c_j \leftarrow$ coefficient of $(e_j \bmod p)$-degree term of $f^{\bmod p}$ for some fixed $p \in \mathcal{P}$;
return $\sum_{j=1}^{t} c_j x^{e_j}$

Observe that p is a good prime if and only if $\#f = \#(f^{\bmod p})$.

Example 3.3.3. *Let $f = 3x + x^6$. Then $f \bmod (x^5 - 1) = 4x$, such that $\#(f^{\bmod 5}) = \#f$ and 5 is a bad prime. $f \bmod (x^3 - 1) = 3x + 1$. As $\#f = \#(f^{\bmod 3})$, we have that and 3 is a good prime.*

If p is not a good prime, then p divides $e_i - e_j$ for some i, j where $1 \leq i \neq j \leq t$. As $\prod_{1 \leq i < j \leq t}(e_i - e_j) < D^{T(T-1)/2}$, this gives us the following bound.

Lemma 3.3.4. *[Step 0, [GS09]] There are at most $\frac{1}{2}T(T-1)\log D$ bad primes.*

We construct $f^{\bmod p}$ for p taken from a set of $\mathcal{O}(T\log^2 D)$ primes $\mathcal{P}$. The algorithm then determines t as $t = \max_{p \in \mathcal{P}}(\#f^{\bmod p})$Once the algorithm determines t it can trivially identify other good primes p via inspection of $f^{\bmod p}$.

Once t is known, we interpolate the *auxillary polynomial*

$$\Lambda(y) = \prod_{i=1}^{t}(y - e_i) \in \mathbb{Z}[y]. \tag{3.2}$$

If p is a good prime, then the exponents of $f^{\bmod p}$ give the distinct values $e_i \bmod p$ for $i \in [t]$ and hence $\Lambda \bmod p$ as well. We can bound the length and thus the height of Λ by $\prod_{i=1}^{t}(1 + e_i) \leq (D+1)^T$. Thus we can compute $f^{\bmod p}$ for $T\log(D+1)$ good primes p and reconstruct Λ from its corresponding images modulo p. We can then factor Λ to obtain the support of f, i.e., $e_1, e_2, \ldots, e_t$. The coefficient c_i corresponding to exponent e_i can be obtained from the coefficient of the degree-$(e_i \bmod p)$ term appearing in any good image $f^{\bmod p}$.

Our bounds on the number of primes needed at each stage is naive, as it does not consider the magnitude of the respective primes; however, using more refined bounds will not improve the soft-oh cost of the algorithm. For completeness we include a pseudocode description of the Garg–Schost method in Algorithm 6.

The query cost entails $\mathcal{O}(T^2 \log D)$ queries of degree $\widetilde{\mathcal{O}}(T^2 \log^2 D)$. The additional costs are as follows:

- *Construction of the primes p*: By Corollary 2.4.3, we can construct the first $\ell \in \widetilde{\mathcal{O}}(T^2 \log D)$ primes in $\widetilde{\mathcal{O}}(T^2 \log D)$ bit operations via the wheel sieve.
- *Construction of the images $\Lambda \bmod p$*: For a prime p_i, $\Lambda \bmod p_i$ may be computed from its linear factors $\{e_i \bmod p : i \in [t]\}$ with $\widetilde{\mathcal{O}}(T)$ $\mathbb{F}_{p_i}$-operations by Theorem 2.1.7, or a bit cost of $\widetilde{\mathcal{O}}(T \log p_i) = \widetilde{\mathcal{O}}(T \log\log D)$. Constructing $\Lambda \bmod p_i$ for $\mathcal{O}(T \log D)$ primes then becomes $\widetilde{\mathcal{O}}(T^2 \log D)$.
- *Construction of the auxillary polynomial Λ, via Chinese remaindering remaindering*: Each of the $\mathcal{O}(T)$ has magnitude at most $(D+1)^T$. Thus the Chinese remaindering will cost $\widetilde{\mathcal{O}}(T \log(D^T)) = \widetilde{\mathcal{O}}(T^2 \log D)$ per Theorem 2.1.6.
- *Factorization of Λ*: In order to obtain the complete factorization we can merely employ Hensel lifting on a good image $f^{\bmod p}$. Per Theorem 2.1.8, and as each of the linear factors $(y - e_i)$ of Λ have height less than D, this costs $\widetilde{\mathcal{O}}(T \log D)$.

This gives a total additional bit cost of $\widetilde{\mathcal{O}}(T^2 \log^2 D)$. Per remark 3.2.4, these costs are dominated by the query cost. This completes the cost analysis of Algorithm 6.

In the case that f is given by a *straight-line program*, Lemma 3.2.2 gives the following Corollary.

Corollary 3.3.5 (Thm. 1, [GS09]). *Algorithm 6 interpolates $f \in \mathsf{R}[x]$ given by a length-L straight-line program with a cost of $\widetilde{\mathcal{O}}(LT^4 \log^2 D)$ R-operations.*

3.3.2 Probabilistic Garg–Schost

In order to expedite the Garg–Schost method, we merely choose a prime p that is good with high probability.

Lemma 3.3.6 (Lemma 2.1, [GR11]). *Let $\mathbb{Z} \ni \lambda \geq \max(21, \frac{5}{3}T(T-1)\ln D)$. Then the probability that a prime p, randomly selected from $\mathcal{P}_{(\lambda..2\lambda]}$, is good, is at least $1/2$.*

Proof. Let $\mathcal{B}$ be the set of bad primes in $(\lambda..2\lambda]$. Per the proof of Lemma 3.3.4, we have that $\prod_{p\in\mathcal{B}} p \leq D^{\frac{1}{2}T(T-1)}$, such that if $\mathcal{B}$ is nonempty, then $\#\mathcal{B} \leq \frac{1}{2}T(T-1)\ln D/\ln\lambda$. By Lemma 2.4.4, the number of primes in $(\lambda..2\lambda]$ is at least $\frac{3}{5}\lambda\ln^{-1}\lambda \geq T(T-1)\ln D\ln^{-1}\lambda \geq 2\#\mathcal{B}$, as desired. □

Algorithm 7: A Monte Carlo variant of the Garg–Schost method

Input: An extended black-box polynomial $f \in \mathsf{R}[x]$; bounds $T \geq \#f$, $D \geq \deg(f)$.
Output: With probability at least $2/3$ a sparse representation of f.

// Construct images of $\Phi \bmod p$ for a set $\mathcal{P}'$ of good primes
1 $\lambda \leftarrow \lceil \max(21,\ \frac{5}{3}T(T-1)\ln D,\ 40\ln(2(D+1)^T+1))\rceil$;
2 $m \leftarrow \lceil 24\log_\lambda(2(D+1)^T+1)\rceil$;
3 $\mathcal{P} \leftarrow$ **GetPrimes**$(\lambda, m; 5/6)$;
4 $t \leftarrow \max_{p\in\mathcal{P}} \#(f^{\bmod p})$;
5 $\mathcal{P}_{\text{good}} \leftarrow \{p \in \mathcal{P} : \#(f^{\bmod p}) = t\}$;
6 **if** $\prod_{p\in\mathcal{P}_{\text{good}}} p \leq 2(D+1)^T+1$ **then return** `fail`;
7 **for** $p \in \mathcal{P}'$ **do** $\Lambda_p \leftarrow \prod_{e\in\text{supp}(f^{\bmod p})}(y-e)$;

// Construct each exponent e_i and pair it with its coefficient
8 Construct $\Lambda \in \mathbb{Z}[x]$ with height at most $(D+1)^T$ from congruences $\{\Lambda_p \bmod p \mid p \in \mathcal{P}_{\text{good}}\}$;
9 Factor $\Lambda = \prod_{j=1}^{t}(y-e_j)$ to get exponents e_j;
10 **for** $j = 1, 2, \ldots, t$ **do** $c_j \leftarrow$ coefficient of $(e_j \bmod p)$-degree term of $f^{\bmod p}$ for some fixed $p \in \mathcal{P}_{\text{good}}$;
11 **return** $\sum_{j=1}^{t} c_j x^{e_j}$

Lemma 3.3.7. *Algorithm 7 succeeds with probability at least $2/3$. It admits a cost of $\widetilde{\mathcal{O}}(T\log D)$ queries of degree $\widetilde{\mathcal{O}}(T^2\log D)$.*

Proof. We first prove correctness. By Lemma 3.3.6, forcing $\lambda \geq 21$ and $\frac{5}{3}T(T-1)\ln D$ guarantees that a randomly selected prime from $(\lambda..2\lambda]$ is a good prime for f with probability at least $1/2$.

By forcing $\lambda \geq 20\ln(2(D+1)^T+1)$, it follows from Corollary 2.4.5 that $(\lambda..2\lambda]$ contains at least $(3/5)\,40\log_\lambda(2(D+1)^T+1) = 24\log_\lambda(2(D+1)^T+1)$ primes, such that **GetPrimes** can select $\lceil 24\log_\lambda(2(D+1)^T+1)\rceil$ distinct primes from $(\lambda..2\lambda]$. By Corollary 2.3.3, and taking $n = \log_\lambda(2(D+1)^T+1), \delta = 1/2$, and $\mu = 1/6$; the probability that $\mathcal{P}$ contains n good primes is at least $5/6$. If $\mathcal{P}$ contains n good primes then we have sufficiently many good primes in order to construct Λ. It follows from the union bound that the algorithm succeeds with probability at least $2/3$.

We now analyze its cost. The query cost is $\widetilde{\mathcal{O}}(T\log D)$ queries of degree $\widetilde{\mathcal{O}}(T\log^2 D)$. The additional bit-operation cost due to lines 8-11 is $\widetilde{\mathcal{O}}(T^2\log^2 D)$, similar to the discussion of the cost of Algorithm 6 at the end of Section 3.3.1. This again is dominated by the query cost, per Remark 3.2.4. □

In the case that f is given by an SLP, this gives us the following cost:

Corollary 3.3.8. *Algorithm 7 interpolates f given by a length-L straight-line program with a cost of $\widetilde{\mathcal{O}}(LT^3\log^2 D)$ ring operations.*

3.3.3 Related work: sparse polynomial identity testing of SLPs

We mention an algorithm, due to Bläser, Hardt, Lipton, and Vishnoi [Bl+09], for the deterministic identity test of a sparse polynomial f, given by an SLP or extended black box, with bounds $T \geq \#f$ and $D \geq \deg(f)$ are known. Their method similarly relies on querying f. The following Lemma is central to their method.

Lemma 3.3.9. *Suppose f is nonzero with $T \geq \#f$ and $D \geq \deg(f)$. Then there are at most $\lfloor (T-1)\log D\rfloor$ primes p such that $f^{\bmod p}$ is zero.*

Proof. Write the sparse representation of f as $f = \sum_i c_i x^{e_i}$. If $f^{\bmod p} = 0$, then $p|(e_1 - e_j)$ for some $j \neq 1$, or else $c_1 x^{e_1}$ is a singleton modulo $(x^p - 1)$, and $f^{\bmod p}$ would have a term $c_1 x^{e_1 \bmod p}$. Thus, if we let $\mathcal{B}$ be the set of primes p satisfying $f^{\bmod P} = 0$; we have that

$$2^{\#\mathcal{B}} \leq \prod_{p\in\mathcal{B}} p \leq \prod_{j=2}^{t}(e_1 - e_j) \leq D^{T-1}.$$

such that $\#\mathcal{B} \leq (T-1)\log D$. □

Lemma 3.3.10. *Procedure **SparsePIT**$(f, f_{\text{sparse}}, D, T)$ is correct and costs $\widetilde{\mathcal{O}}(T\log D)$ queries of degree $\widetilde{\mathcal{O}}(T\log D)$, and an additional $\widetilde{\mathcal{O}}(T^2\log^2 D)$ bit operations and $\widetilde{\mathcal{O}}(T^2\log D)$ ring operations.*

Procedure SparsePIT($f, f_{\text{sparse}}, D, T$)

Input: $f \in \mathsf{R}[x]$, an extended black-box polynomial; $f_{\text{sparse}} \in \mathsf{R}[x]$, a sparse polynomial; $D \geq \deg(f), \deg(f_{\text{sparse}})$; $T \geq \#f, \#f_{\text{sparse}}$
Result: Accepts if and only if $f = f_{\text{sparse}}$.

1 $\mathcal{P} \leftarrow$ set of first $\lfloor (2T-1)\log D \rfloor + 1$ primes ;
2 **for** $p \in \mathcal{P}$ **do**
3 $\quad$ **if** $(f - f_{\text{sparse}})^{\bmod p} = 0$ **then reject**;
4 **accept**;

Proof. Correctness follows from Lemma 3.3.9. By the Prime Number Theorem (Thm. 2.4.2), we have that all the primes p_i computed on line 1 satisfy $p_i \in \widetilde{\mathcal{O}}(T \log D)$. We can construct all such primes in $\widetilde{\mathcal{O}}(T \log D)$ bit operations via the wheel sieve, by Theorem 2.4.1. The construction of each image $f^{\bmod p_i}$ entails a query of degree $\widetilde{\mathcal{O}}(T \log D)$. Constructing $f_{\text{sparse}}^{\bmod p_i}$ entails $\widetilde{\mathcal{O}}(T \log D)$ bit operations and $\widetilde{\mathcal{O}}(T)$ ring operations. Taking the difference $(f - f_{\text{sparse}})^{\bmod p_i} = f^{\bmod p_i} - f_{\text{sparse}}^{\bmod p_i}$ entails an additional $\mathcal{O}(T)$ ring operations. Accounting for all $\widetilde{\mathcal{O}}(T \log D)$ primes completes the proof. □

Corollary 3.3.11. ***SparsePIT*** *costs $\widetilde{\mathcal{O}}(LT^2 \log^2 D)$ ring operations for f given by a straight-line program of length L.*

Procedure **SparsePIT** allows for any Monte Carlo interpolation procedure for an extended black box, and taking bounds D and T as inputs, to be made Las Vegas. We merely run the Monte Carlo sparse interpolation algorithm repeatedly until the polynomial identity test given by **SparsePIT** verifies the output to be correct. Algorithm 8 gives pseudocode for the approach. In the case that the Monte Carlo procedure **SparseInterpProc** has cost $\mathcal{O}(\Psi(D,T))$, the expected ring operation cost of Algorithm 8 will be $\widetilde{\mathcal{O}}(\max(T^2 \log^2 D, \Psi(D,T)))$.

Algorithm 8: Las Vegas sparse interpolation of SLPs

Input: **SparseInterpProc**, a Monte Carlo algorithm that interpolates a sparse polynomial given by an extended black-box with probability at least $2/3$; $D \geq \deg(f)$; $T \geq \#f$
Output: A sparse representation of f.

1 **while** `true` **do**
2 $\quad f_{\text{sparse}} \leftarrow$ output of **SparseInterpProc** with inputs f, D, and T;
3 $\quad$ **if** $\#f_{\text{sparse}} \leq T$, *and* ***SparsePIT***$(f, f_{\text{sparse}}, D, T)$ *accepts* **then return** f_{sparse};

3.3.4 The Giesbrecht–Roche "diversified" method

In [GR11], the authors present a probabilistic algorithm with improved cost over the Garg–Schost algorithm for the case that R is an integral domain with order exceeding the degree of the interpolant f by some constant factor. Their strategy is to construct, for $f = \sum_{i=1}^{t} c_i x^{e_i}$, shifted queries of the form

$$f(\alpha x) \bmod p = \sum_{i=1}^{t} c_i \alpha^{e_i} x^{e_i \bmod p}. \tag{3.3}$$

for a set of good primes p and a fixed choice of $\alpha \in \mathbb{F}_q$, such that the nonzero coefficients of $f(\alpha x)$ are, with high probability, distinct. This technique is called **diversification**.

Definition 3.3.12. *We say $f \in \mathsf{R}[x]$ is* **diverse** *if its nonzero coefficients are distinct. We say $\alpha \in \mathsf{R}$ (or a ring extension of R)* **diversifies** *f if $f(\alpha x)$ is diverse.*

Using diversification, we may avoid the construction of the polynomial Λ (3.2). Instead, by collecting terms from the images $f(\alpha x)^{\bmod p}$ according to their coefficient, we obtain lists of congruences each corresponding to a distinct term cx^e of $f(\alpha x)$. By probabilistically selecting $\mathcal{O}(\log D)$ good primes $p \in \widetilde{\mathcal{O}}(T^2 \log D)$, one can reconstruct the exponents of f via Chinese remaindering on these sets of congruences. In order to probabilistically select an α that diversifies f, we prove the following using the DLSZ Lemma (Thm. 2.5.7). The proof mimics that of Theorem 3.1 of [GR11].

Lemma 3.3.13. *Let R be a ring and $\mathcal{S} \subseteq \mathsf{R}$ a regular-difference set of cardinality at least $2D$. Let $m = \lceil \log(T(T-1)\mu^{-1}) - 1 \rceil$, and choose $\alpha_1, \ldots, \alpha_m \in \mathcal{S}$ independently and uniformly at random. Then*

$$\boldsymbol{f}(x) = (f(\alpha_1 x), \ldots, f(\alpha_m x)) \in \mathsf{R}^m[x]$$

is diverse with probability at least $1 - \mu$.

Proof. For $f = \sum_{i=1}^{t} c_i x^{e_i}$, the term of degree e_i in $\boldsymbol{f}$ will have coefficient

$$(c_i \alpha_1^{e_i}, \ldots, c_i \alpha_m^{e_i}),$$

such that the ith and jth terms will share a coefficient if and only if $(c_i \alpha_k^{e_i} - c_j \alpha_k^{e_j})$ for all $k \in [m]$. By the DLSZ Lemma this occurs with probability at most $2^{-m} \leq \mu / (\frac{1}{2} T(T-1))$. Taking the union bound over the at-most $\frac{1}{2} T(T-1)$ pairs (i, j), $i \leq j \in [t]$, completes the proof. □

Corollary 3.3.14 (Thm. 3.1, [GR11])**.** *Let q be a prime power and let $s \geq \log_q \left(\frac{1}{2} T(T-1) D \mu^{-1} \right)$. If α is randomly chosen from $\mathbb{F}_{q^s}$, then α diversifies f with probability at least $1 - \mu$.*

Algorithm 9: Giesbrecht–Roche diversified sparse interpolation

Input: An extended black-box polynomial $f \in \mathbb{F}_q[x]$; $D \geq \deg(f)$; $T \geq \#f$
Output: With probability at least $2/3$, a sparse representation of f

// Construct good images f_p
$\lambda \leftarrow \lceil \max(21,\ \frac{5}{3}T(T-1)\ln D,\ 30\ln D) \rceil$;
$\mathcal{P} \leftarrow$ **GetPrimes**$(\lambda, 18\log_\lambda D; 1/9)$;
$t \leftarrow \max_{p\in\mathcal{P}} \left(\#f \bmod p\right)$;
$\mathcal{P}_{\text{good}} \leftarrow \{p \in \mathcal{P} \mid \#f \bmod p = t\}$;

// Diversify f
$k \leftarrow \lceil \frac{9}{2}T(T-1) \rceil$;
$s \leftarrow \lceil \log_q(2D) \rceil$;
Choose $\alpha_1, \ldots, \alpha_k \in \mathbb{F}_{q^s}$ independently and uniformly at random;
$\boldsymbol{f}_p \leftarrow (f(\alpha_1 x) \bmod p, \ldots, f(\alpha_k x) \bmod p)$, for some fixed $p \in \mathcal{P}_{\text{good}}$;
$\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t \leftarrow \text{coeffs}(\boldsymbol{f}_p)$;
if $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t$ *are not distinct* **then return** `fail`;

// Construct congruences for each exponent
for $p \in \mathcal{P}'$ **do**
  $\boldsymbol{f}_p \leftarrow (f(\alpha_1 x) \bmod p, \ldots, f(\alpha_k x) \bmod p)$;
  if $\text{coeffs}(\boldsymbol{f}_p) \neq \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t\}$ **then return** `fail`;
  for $i \leftarrow 1$ **to** t **do** $e_{ip} \leftarrow$ exponent of term with coefficient $\boldsymbol{c}_i$ in $\boldsymbol{f}_p$;

// Construct exponents via Chinese remaindering
for $i \leftarrow 1$ **to** t **do** $e_i \leftarrow$ solution in $[0..D]$ to congruences $\{e_i \equiv e_{ip} \pmod p : p \in_{\text{good}}\}$;
return $\sum_{i=1}^{t} c_{i1}\alpha_1^{-e_i} x^{e_i}$;

Theorem 3.3.15. *Let $f \in \mathbb{F}_q[x]$ be an extended black-box polynomial with bounds $D \geq \deg(f)$ and $T \geq \#f$. Given f, D, and T, Algorithm 9 interpolates f with probability at least $2/3$. It costs $\widetilde{\mathcal{O}}(\log D)$ $\mathbb{F}_{q^s}$-shifted queries of degree $\widetilde{\mathcal{O}}(T^2 \log D)$, where $s \in \mathcal{O}(1 + \log_q D)$*

Proof. We first prove correctness. λ is chosen to be at least 21 and $\frac{5}{3}T(T-1)\ln D$, such that by Lemma 3.3.6, a randomly chosen prime from $(\lambda, 2\lambda]$ is a good prime with probability at least $1/2$. By Corollary 2.4.5, by choosing $\lambda \geq 30\ln D$, we guarantee that $(\lambda..2\lambda]$ contains at least $18\log_\lambda D$ primes, such that **GetPrimes** produces $\lceil 18\log_\lambda D \rceil$ from $(\lambda..2\lambda]$. By Corollary 2.3.3, taking $m = 18\log_\lambda D, n = \log_\lambda D$, $\delta = 1/2$, and $\mu = 1/9$ the probability that fewer than $\log_\lambda D$ primes in $\mathcal{P}$ are good is less than $1/9$. Per Lemma 3.3.13, $\boldsymbol{f} = (f(\alpha_1 x), \ldots, f(\alpha_k x)) \in \mathsf{R}^k[x]$ is

diverse with probability at least $1/9$. As these comprise the probabilistic steps of the algorithm, by the union bound we succeed with probability at least $2/3$ as desired.

Per Theorem 2.1.6, the cost of constructing the exponents e_i is softly-linear in the combined bit size of the exponents e_i, i.e., $\widetilde{\mathcal{O}}(T \log D)$ bit operations. Computing $\alpha_1^{-e_i}$ for $i \in [t]$ costs $\widetilde{\mathcal{O}}(T \log D)$ $\mathbb{F}_{q^s}$-operations via a square-and-multiply approach, or a bit cost of $\widetilde{\mathcal{O}}(sT \log D \log q) = \widetilde{\mathcal{O}}(T \log D(\log D + \log q)$. We can construct such a field extension $\mathbb{F}_{q^s}$ by the first algorithm of Theorem 2.1.5, with a bit cost $\widetilde{\mathcal{O}}(s^2 \log q + s \log^2 q) \subseteq \widetilde{\mathcal{O}}(\log^3 D)$.

These costs are dominated by the cost of the $\widetilde{\mathcal{O}}(\log D)$ shifted queries of degree $\widetilde{\mathcal{O}}(T^2 \log D)$ over $\mathbb{F}_{q^s}$, which per Remark 3.2.4 have a bit cost of at least

$$\widetilde{\mathcal{O}}(T^2 \log^2 Ds \log q) = \widetilde{\mathcal{O}}(T^2 \log D(\log D + \log q)).$$

□

This algorithm can be made Las Vegas by the sparse PIT of Blaser et al. In the case that f is given by an SLP this gives the following cost.

Corollary 3.3.16. *There exists a Las Vegas algorithm that, given a length-L SLP computing $f \in \mathbb{F}_q$, and $T \geq \#f$ and $D \geq \deg(f)$, interpolates f with an expected cost of $\widetilde{\mathcal{O}}(LT \log^2 D(\log D + \log q))$ bit operations.*

3.4 Iterative sparse interpolation

In this section we present sparse interpolation algorithm for an extended black-box polynomial f over an arbitrary ring R with identity, with cost softly linear in T. This algorithm was first described in [AGR13], and is joint work with Daniel S. Roche and Mark Giesbrecht. This approach is unique amongst the algorithms in this section for constructing queries $f^{\bmod pq}$ of degree pq, a product of two primes. We will fix a prime p for which most of the exponents of f are singletons modulo p. We choose a list of primes q which will give a set of congruences that allows us to construct the singleton exponents of f.

The algorithm maintains a sparse polynomial f_{sparse} "approximating" f, where, after the end of the ith iteration and with high probability $\#(f - f_{\text{sparse}}) \leq 2^{-i}T$. After $\lceil \log T \rceil + 1$ iterations, this gives us $f_{\text{sparse}} = f$. In each iteration, we add to f_{sparse} over half of the remaining terms of $f - f_{\text{sparse}}$, plus possibly some additional *deceptive* terms constructed due to collisions of terms in modular images of $f - f_{\text{sparse}}$. As such we call this approach **iterative sparse interpolation**. Throughout Section 3.4, we will let $g = f - f_{\text{sparse}}$.

3.4.1 Allowing for some collisions with σ-support primes

Instead of choosing a prime p modulo which all the exponents of g remain distinct, we choose a prime for which *most* of the exponents remain distinct.

Definition 3.4.1. *For $g \in \mathsf{R}[x]$ and any prime p, we let $\mathcal{C}_g(p)$ denote the number of terms of g that are in collisions modulo $(x^p - 1)$.*

*For any proportion $\sigma \in (0, 1]$, we say p is a σ-**support prime** for g if at least $\sigma(\#g)$ exponents of g are singletons modulo p. That is, if $\mathcal{C}_g(p) \leq (1 - \sigma)\#g$.*

Example 3.4.2. *For the polynomial $g = 1 + x^5 + x^7 + x^{10}$, we have $g \bmod (x^2 - 1) = 2 + 2x$ and $g \bmod (x^5 - 1) = 3 + x^7$. One can check that $\mathcal{C}_g(2) = 4$ as 1 collides with x^{10} and x^5 collides with x^7 modulo $x^2 - 1$. $\mathcal{C}_g(5) = 3$, as $1, x^5$, and x^{10} all collide modulo $x^5 - 1$, but the term x^7 is a singleton.*

A 1-support prime is precisely a good prime. The following lemma allows us to find σ-support primes.

Lemma 3.4.3. *Let $g \in \mathsf{R}[x]$ be T-sparse with degree at most D, and fix $\sigma, \mu \in (0, 1)$. Let $\lambda \in \mathbb{Z}$ where $\lambda \geq \max(21, \lceil \frac{5}{3}(T - 1)(1 - \sigma)^{-1}\mu^{-1} \ln D \rceil)$. Then p chosen uniformly at random from the set of primes in $(\lambda..2\lambda]$ is an σ-support prime with probability at least $1 - \mu$.*

Proof. Write $g = \sum_{i=1}^{t} c_i x^{e_i}$, where $t \leq T$ and let $\mathcal{B}$ denote the set of "bad" primes $p \in (\lambda..2\lambda]$ that are not σ-support primes. For every $p \in \mathcal{B}$, more than a proportion of $(1 - \sigma)$ of the terms of e are non-singletons modulo $(x^p - 1)$. For every non-singleton $e_i \in \operatorname{supp}(f)$, there exists $e_j \in \operatorname{supp}(g)$, $e_j \neq e_i$, such that p divides $e_i - e_j$. If $\mathcal{B}$ is nonempty it follows that

$$\lambda^{\#\mathcal{B}(1-\sigma)t} < \prod_{p \in \mathcal{B}} p^{(1-\sigma)t} \leq \prod_{i=1}^{t} \prod_{\substack{j=1 \\ j \neq i}}^{t} (e_i - e_j) < D^{T(T-1)},$$

so $\#\mathcal{B} < (T - 1)(1 - \sigma)^{-1} \ln D \ln^{-1} \lambda$. By Lemma 2.4.4, the number of primes in $(\lambda..2\lambda]$ exceeds $\frac{3}{5}\lambda / \ln \lambda$. As

$$\frac{\#\mathcal{B}}{\frac{3}{5}\lambda \ln^{-1} \lambda} < \frac{(T - 1)(1 - \sigma)^{-1} \ln D \ln^{-1} \lambda}{\frac{3}{5}\lambda \ln^{-1} \lambda} = \frac{5(T - 1)(1 - \sigma)^{-1} \ln D}{3\lambda} \leq \mu^{-1},$$

this completes the proof. □

We will look for $5/8$-support primes. In [AGR13] these were referred to as *ok primes*. We choose $\lambda = \lceil \max(21, \frac{160}{9}(T - 1) \ln D) \rceil$ such that a prime p chosen uniformly at random from $(\lambda..2\lambda]$ is an $13/16$-support prime with probability at least $1/2$.

In order to boost the probability of success, we will choose primes p and choose the best candidate from among them. A natural choice is to take a p which maximizes $\#(g^{\bmod p})$. However, as the following example shows, maximizing $\#(g \bmod p)$ does not necessarily minimize $\mathcal{C}_g(p)$, the number of terms that collide modulo $(x^p - 1)$.

Example 3.4.4. *Let*

$$g = 1 + x + x^4 - 2x^{13}.$$

We have

$$g^{\bmod 2} = 2 - x, \quad \textit{and} \quad g^{\bmod 3} = 1.$$

While $g^{\bmod 2}$ has more terms than $g^{\bmod 3}$, we see that $\mathcal{C}_g(2) = 4$ whereas $\mathcal{C}_g(3) = 3$.

While we cannot determine the prime p for which $g \bmod (x^p - 1)$ has minimally many colliding terms, the following lemma shows that choosing the prime p that maximizes $\#(g^{\bmod p})$ is within a factor 2 of optimal.

Lemma 3.4.5. *Suppose that $g \in \mathsf{R}[x]$ has t terms and $g \bmod p$ has s terms. Then $t - s \leq \mathcal{C}_g(p) \leq 2(t - s)$.*

Proof. To prove the lower bound, note that $t - \mathcal{C}_g(p)$ exponents of g will not collide modulo p, and so $g \bmod (x^p - 1)$ has at least $t - \mathcal{C}_g(p)$ terms.

We now prove the upper bound. There are $\mathcal{C}_g(p)$ terms of g that collide modulo $(x^p - 1)$. Let h be the $\mathcal{C}_g(p)$-sparse polynomial comprised of those terms of g. As each term of h collides with at least one other term of h, $h \bmod p$ has sparsity at most $\mathcal{C}_g(p)/2$. Since all of the terms of $g - h$ are singletons, $(g - h) \bmod p$ has sparsity exactly $t - \mathcal{C}_g(p)$. It follows that $g \bmod p$ has sparsity at most $t - \mathcal{C}_g(p) + \mathcal{C}_g(p)/2 = t - \mathcal{C}_g(p)/2$. That is, $s \leq t - \mathcal{C}_g(p)$, and so $\mathcal{C}_g(p) \leq 2(t - s)$. □

Corollary 3.4.6. *Suppose $g \bmod p$ has sparsity s_p, and $g \bmod q$ has sparsity s_q, where $s_p \geq s_q$. Then $\mathcal{C}_g(p) \leq 2\mathcal{C}_g(q)$.*

Proof.

$$\begin{aligned} \mathcal{C}_g(p) &\leq 2(t - s_p) && \textit{by the second inequality of Lemma 3.4.5,} \\ &\leq 2(t - s_q) && \textit{since } s_p \geq s_q, \\ &\leq 2\mathcal{C}_g(q) && \textit{by the first inequality of Lemma 3.4.5.} \end{aligned}$$

□

Observe that Examples 3.4.2 and 3.4.4 agree with Corollary 3.4.6. By Corollary 3.4.6, if $\{p_1, \ldots, p_m\}$ contains at least one $13/16$-support prime, then the prime p_i for which $\#g \bmod p_i$ is maximal will necessarily be a $5/8$-support prime.

3.4.2 Deceptive terms

Once we have probabilistically identified a $5/8$-support prime p for $g = f - f_{\text{sparse}}$, we construct images of the form $g^{\bmod pq}$ for $q = 2, 3, 5, \ldots$, until we have sufficient information to construct the exponents of what we hope are the singletons appearing in $g^{\bmod p}$. Observe that if a term cx^e of g is a singleton modulo $(x^p - 1)$, then $g^{\bmod pq}$ will have a unique term of the form cx^{e_q} where $e_q \equiv e \pmod p$. The exponent e_q satisfies $e_q \equiv e \pmod q$ as well, so that we can use these congruences, for a sufficiently large set of primes q, to reconstruct e via Chinese remaindering.

If there is a collision of terms, however, the terms may interact in a way that fools us into constructing a **deceptive term**: a term constructed from the images of g that is not a term of g.

Example 3.4.7. *Let*

$$g(x) = 1 + x + x^2 + x^3 + x^{11+4} - x^{14\cdot 11+4} - x^{15\cdot 11+4},$$

and suppose we are given an extended black for g, and bounds $D = 200 \geq \deg(g)$ and $T = 7 \geq \#g$. Let $p = 11$, and suppose we construct images

$$\begin{aligned} g^{\bmod 11} &= 1 + x + x^2 + x^3 - x^4, \\ g^{\bmod 22} &= 1 + x + x^2 + x^3 - x^{15}, \\ g^{\bmod 33} &= 1 + x + x^2 + x^3 - x^{26}, \\ g^{\bmod 55} &= 1 + x + x^2 + x^3 - x^{48}, \\ g^{\bmod 77} &= 1 + x + x^2 + x^3 - x^{15}. \end{aligned}$$

Collecting terms amongst the images with like degree modulo 11, and reconstructing exponents via Chinese remaindering on the resulting congruences modulo $q = 2, 3, 5, 7$, we get the terms $1, x, x^2$, and x^3, and in addition an additional deceptive term $-x^{113}$ not appearing in g.

In the case that a collision is only comprised of two terms of g, then we will be able to detect if it occurs. Namely, if cx^e and $c'x^{e'}$ collide modulo $(x^p - 1)$, then we will choose a prime q such that $e \equiv e' \pmod q$. For such a prime q there will be two terms of degree congruent to e modulo p. In which case it is clear that a collision occured in $g^{\bmod p}$ at degree $e \bmod p$, and we know not to construct a term in that case. In practise, one may implement a *prefix dictionary* (Section 2.2) in order to store a term of cx^d of $g^{\bmod pq}$ with key $((d \bmod p), i, d)$ and value c. The cost of accessing and traversing all key-value pairs is softly linear in the bit sizes of the key-value entries.

3.4.3 Interpolation

Procedure **IterativeSI** (i.e., "Iterative Sparse Interpolation") describes the interpolation procedure, which iteratively interpolates $g = f - f_{\text{sparse}}$. We construct every singleton of g modulo

$(x^p - 1)$, and potentially $\mathcal{C}_g(p)/3$ deceptive terms, each resulting from a collision of three or more terms of g. Thus, if p is an $^5/_8$-support prime, we will construct a sum of terms f'_{sparse} such that $\#(g - f'_{\text{sparse}}) \leq \frac{4}{3}\mathcal{C}_g(p) = \frac{1}{2}\#(g)$. We accordingly replace f_{sparse} with $f_{\text{sparse}} + f'_{\text{sparse}}$, and replace bound $T_g \geq \#(f - f_{\text{sparse}})$ with $\lfloor T_g/2 \rfloor$. We continue in this manner until $T_g = 0$, at which point $f = f_{\text{sparse}}$, provided the probabilistic steps of the algorithm succeeded.

Procedure IterativeSI(f, D, T)

Input: An extended black-box polynomial $f \in \mathsf{R}[x]$; $D \geq \deg(f)$, $T \geq \#f$.
Output: With probability at least $2/3$, a sparse representation of f

```
   // Initialize values
1  T_g ← T;
2  f_sparse ← 0 ∈ R[x], a sparse polynomial;
3  μ ← 1/(3⌈log T + 1⌉);
4  Q ← the first ⌈log D⌉ primes, generated by the wheel sieve;

5  while T_g > 0 do
       // Build images
6      λ ← max(21, ⌈(320/9)(T − 1) ln D⌉);
7      m ← log(2μ^{-1});
8      p_1, ..., p_m ← GetPrimes(m, λ; μ/2);
9      for i ← 1 to m do Compute (f − f_sparse) mod p_i;
10     p ← a prime p_i for which #(f − f_sparse) mod p_i is maximal;
11     for q ∈ Q do Construct (f − f_sparse) mod pq;

       // Reconstruct exponents via Chinese remaindering
12     f'_sparse ← 0 ∈ R[x], a sparse polynomial;
13     for d ∈ supp(f mod p) do
14         if for each q ∈ Q, f mod pq has a unique nonzero term of degree d_q ≡ d (mod p) then
15             e ← solution to congruences (e ≡ d_q)_{q∈Q};
16             c ← coefficient of degree-(e mod p) term of (f − f_sparse) mod p;
17             f'_sparse ← f'_sparse + c x^e;
18     f_sparse ← f_sparse + f'_sparse;
19     T_g ← ⌊T_g/2⌋;
20     if #f_sparse > T_g + T then return fail;
21 return f_sparse
```

Theorem 3.4.8. ***IterativeSI****(f, D, T) interpolates f with probability at least $2/3$, and a cost of $\mathcal{O}(\log D \log T)$ queries of degree $\widetilde{\mathcal{O}}(T \log^2 D)$.*

Proof. We first prove correctness. Since we initialize T_g to T and decrease T_g by a factor of at least 2 each iteration, we will have $T_g \leq 1$ after $\lfloor \log T \rfloor$ iterations. It follows that there are at most $\lfloor \log T + 1 \rfloor$ iterations of the outer while loop. Thus, if every iteration succeeds with probability at least $1 - \mu$, where $\mu = \frac{1}{3}\lceil \log T + 1 \rceil^{-1}$, then by the union bound the algorithm succeeds with probability at least $2/3$.

The only probabilistic step in each such iteration is the selection of an $5/8$-support prime p. **GetPrimes**$(m, \lambda; \mu/2)$ produces a set $\mathcal{P}$ comprising m primes from $(\lambda..2\lambda]$, or all primes from $(\lambda..2\lambda]$ if the interval contains fewer than m primes. The procedure produces these primes with probability at least $1 - \mu/2$. By the choice of λ and Lemma 3.4.3, a prime chosen from $(\lambda..2\lambda]$ is a $13/16$-support prime with probability at least $1/2$. It follows that the probability that $\mathcal{P}$ contains a $13/16$-support prime is at least $1 - \mu/2$. If this holds, then by selecting $p \in \mathcal{P}$ to maximize $(f - f_{\text{sparse}}) \bmod p$, p is an $5/8$-support prime by Corollary 3.4.6. Thus, by the union bound, a single iteration succeeds with probability at least $1 - \mu$, as desired.

We now analyze the cost of one iteration of the outer for loop. The query cost is $\widetilde{\mathcal{O}}(\log D)$ queries of degree $\widetilde{\mathcal{O}}(T \log^2 D)$. The cost of constructing m primes in $(\lambda..2\lambda]$ via **GetPrimes** is $\widetilde{\mathcal{O}}(m \cdot \text{polylog}(\lambda)) \subseteq \widetilde{\mathcal{O}}(m \cdot \text{polylog}(T \log D))$ bit operations, per Propositon 2.4.8. By the test on line 20, we have as a loop invariant that $\#f_{\text{sparse}} \leq 2T$.

Chinese remaindering of each set of congruences, per Theorem 2.1.6, costs $\widetilde{\mathcal{O}}(\log D)$ bit operations, for a total of $\widetilde{\mathcal{O}}(T \log D)$. Per Remark 3.2.4 the cost of constructing images of $g = f - f_{\text{sparse}}$ is dominated by the cost of querying f, which is $\widetilde{\mathcal{O}}(\log T + \log D)$ queries of degree $\widetilde{\mathcal{O}}(T \log^2 D)$, or a bit and R operation cost of $\widetilde{\mathcal{O}}(T \log^3 D)$.

As accounting for all $\mathcal{O}(\log T)$ iterations does not affect the soft-oh cost, this completes the proof. □

Corollary 3.4.9. ***IterativeSI****(f, D, T) interpolates f given by a length-L SLP, with a $\widetilde{\mathcal{O}}(LT \log^3 D)$ ring operation cost. If in addition f is over $\mathbb{F}_q$,* ***IterativeSI****(f, D, T) costs $\widetilde{\mathcal{O}}(LT \log^3 D \log q)$ bit operations.*

3.5 Majority-rule sparse interpolation over finite fields

In this section we present another sparse interpolation algorithm that improves over that of the previous section, in the case that f is over an integral domain of order exceeding D by some constant factor. This algorithm appears in [AGR14] for f over a finite field and was joint work with Mark Giesbrecht and Daniel S. Roche. In the original presentation of this algorithm, we

interpolated f in an iterative fashion similar to that of iterative sparse interpolation, whereby we maintain a sparse polynomial f_{sparse} and attempt to construct terms of the difference $g = f - f_{\text{sparse}}$. Here we will present a variant of the algorithm that constructs all the terms in a single batch. This was a strategy we used later in a multivariate interpolation algorithm presented in Section 5.3.

In **IterativeSI**, we distinguish terms by their degree reduced modulo p for an appropriate choice of a prime p. We were unable to naively apply the diversification technique of Giesbrecht and Roche because we choose a prime p that is not necessarily a good prime for g. In the algorithm we will present in this section, we use a generalization of diversification in order to probabilistically distinguish not only images of distinct terms of f, but also images of *collisions* of terms of f that appear in a set of ℓ queries $f^{\bmod p_i}$ for primes $p_i \in \mathcal{O}(T \log D)$. This notion of diversification will allow us to collect terms belonging to the images f_i according to their preimage, by collecting terms with like coefficients. In this way we can identify a pair of terms τ of f_i and τ' of f_j as being images of the same term of f, or the same sum of terms of f.

Every image will "vote" on whether such a collection corresponds to a single term of f, or a collision. Essentially f_i votes for a collection if it has a term belonging to that collection. We will construct terms for every collection of size at least $\ell/2$, i.e., those collections for which at least half of terms of f vote in favour of. As such, we will call our algorithm *majority-rule* sparse interpolation.

3.5.1 Selecting a set of primes of f

For the purposes of Section 3.5.1 it suffices to consider $f \in \mathsf{R}[x]$. We first aim to construct a list of distinct primes $(p_i)_{1 \le i \le \ell}$ and corresponding images $f_i = f^{\bmod p_i}$ such that the following conditions hold:

(3.5.i) Every term of f appears as a singleton in over half of the images $f^{\bmod p_i}$.

(3.5.ii) Any half of the primes p_i have a product exceeding D.

We will only guarantee (3.5.i) probabilistically. If either (3.5.i) or (3.5.ii) hold, given that any two terms differ by degree at most D, any fixed sum of two or more terms of f must occur in fewer than half of the images f_i. If both hold, then for any term $\tau = cx^e \in f$, the images $f^{\bmod p_i}$ for which τ is a singleton will give us $\boldsymbol{e} \bmod p_i$, and these congruences will be sufficient in order to reconstruct the exponent e. In order to construct such a set of primes we use the following lemma:

Lemma 3.5.1. *Let $f(x) \in \mathsf{R}[x]$ be T-sparse with degree at most $D \ge 1$, τ a fixed term of f, and $\mathbb{Z} \ni \lambda \ge \max(21, \lceil (T-1)\mu^{-1} \ln D \rceil)$. Then for a prime chosen uniformly at random from $(\lambda..2\lambda]$, τ is in a collision modulo $(x^p - 1)$ with probability at most $1 - \mu$.*

Proof. Write $f = \sum_{i=1}^{t} c_i x^{e_i}$ and suppose τ is the ith term $c_i x^{e_i}$. Let $\mathcal{B} \subset \mathcal{P}_{(\lambda..2\lambda]}$ be the set of bad primes p in $(\lambda, 2\lambda]$ for which τ collides in the image $f^{\bmod p}$. If τ collides with the jth term $c_j x^{e_j}$ in $f^{\bmod p}$, $j \neq i \in [t]$, then p divides $e_i - e_j$, and

$$\lambda^{\#\mathcal{B}} < \prod_{p \in \mathcal{B}} p \leq \prod_{\substack{j=1 \\ j \neq i}}^{t} |e_i - e_j| \leq D^{T-1}.$$

Thus, if $\mathcal{B}$ is nonempty, $\#\mathcal{B} < (T-1) \ln D \ln^{-1} \lambda$. By Lemma 2.4.4, the number of primes in $(\lambda..2\lambda]$ is at least $\frac{3}{5}\lambda \ln^{-1} \lambda$, and so

$$\frac{\#\mathcal{B}}{\#\mathcal{P}_{(\lambda,2\lambda]}} \leq \frac{(T-1) \ln D \ln^{-1} \lambda}{\frac{3}{5}\lambda \ln^{-1} \lambda} \leq \frac{5(T-1) \ln D}{3\lambda} = \mu.$$

□

Corollary 3.5.2. *Let f be as in Lemma 3.5.1. Suppose $\lambda \in \mathbb{Z}, \lambda \geq \max(21, \lceil \frac{20}{3}(T-1) \ln D \rceil)$, and*

$$\ell \geq \min\left(8 \ln(T\mu^{-1}),\ \ \#\mathcal{P}_{(\lambda..2\lambda]}\right).$$

Then for ℓ distinct primes $p_1, \ldots, p_\ell$ chosen independently and uniformly at random from $\mathcal{P}_{(\lambda..2\lambda]}$, every term τ of f collides in fewer than half of the images $f^{\bmod p_i}, i \in [\ell]$, with probability at least $1 - \mu$.

Proof. By Lemma 3.5.1, τ collides in an image $f^{\bmod p_i}$ with probability less than $1/4$. In the case that $\ell = \#\mathcal{P}_{(\lambda..2\lambda]}$, every term τ of f will necessarily collide in fewer than $1/4$ of the images f_i.

Consider the case then that $\ell < \#\mathcal{P}_{(\lambda..2\lambda]}$. Fix a term τ of f, and let X_i be the indicator variable that τ collides modulo $(x^{p_i} - 1)$. Observe that the random variables $X_i, i \in [\ell]$ are negatively correlated, and that $\mathbb{E}[X_i] < 1/4$. It follows from Hoeffding's inequality (Thm. 2.3.4) that

$$\Pr\left[\sum_{i=1}^{\ell} X_i \leq \ell/2\right] \leq \exp(-\ell/8) \leq T^{-1}\mu.$$

Taking the union bound over the T terms of f completes the proof. □

3.5.2 Generalized diversification

Here we now suppose $f \in \mathbb{F}_q[x]$. Once we have generated a list of primes $(p_1, \ldots, p_\ell)$ satisfying properties (3.5.i) and (3.5.ii), and their corresponding images $f_i = f^{\bmod p_i}, i \in [\ell]$, the remaining

challenge is to collect terms amongst the images f_i according to their preimage. In order to accomplish this, we will construct ℓm images of the form

$$f_{ij} = f(\alpha_j x)^{\bmod p_i}, \quad i \in [\ell], j \in [m],$$

where α_j is chosen from a sufficiently large field extension $\mathbb{F}_q$. Any term of an image f_i is an image of either a single term of f or a sum of multiple terms of f. To analyze our situation, consider the bivariate polynomials

$$f(yx) \bmod (x^{p_i} - 1) = \sigma_{i,0} + \sigma_{i,1}x + \cdots + \sigma_{i,p_i-1}x^{p_i-1},$$

where each $\sigma_{i,u} \in \mathsf{R}[y]$ is the sum of terms of $f(y)$ with degrees congruent to $u \pmod p$. Each $\sigma_{i,u}$ has between 0 and T terms and degree at most D in y. The nonzero $\sigma_{i,u}(x)$ are precisely the singletons and collisions of f under the reductions modulo $(x^{p_i}-1)$, $i \in [\ell]$. For a choice of $\alpha \in \mathsf{R}$, computing $f(\alpha x)^{\bmod p_i}$ gives a univariate polynomial whose coefficient of degree u is $\sigma_{i,u}(\alpha)$. If a term τ of f is a singleton modulo $(x^{p_i}-1)$, then $\tau(y) = \sigma_{i,u}(y)$ for some u. If τ is also a singleton modulo $(x^{p_k}-1)$, then there exists some v such that $\sigma_{i,u} = \sigma_{k,v}$. Obviously, for any $\alpha \in \mathsf{R}$, we then have $\sigma_{i,u}(\alpha) = \sigma_{k,v}(\alpha)$.

One problem is that we may have $\sigma_{i,u}(\alpha) = \sigma_{k,v}(\alpha)$, but $\sigma_{i,u} \neq \sigma_{k,v}$. That is, we have that $f(\alpha x)^{\bmod p_i}$ and $f(\alpha x)^{\bmod p_k}$ have terms sharing the same coefficient but with different preimages. We call this event a **deception**, since it may fool our algorithm into constructing a deceptive term that is not a term of f. If α_j is such that $\sigma_{i,u}(\alpha_j) \neq \sigma_{k,v}(\alpha_j)$, then α_j distinguishes the pair $\sigma_{i,u}, \sigma_{k,v}$ as having distinct preimages.

Definition 3.5.3. *Let $\Phi_1, \ldots, \Phi_\ell : \mathsf{R}[x] \mapsto \mathsf{R}[x]$ be sparsity-preserving maps. We say $\alpha_1, \ldots, \alpha_m$ is a* **diversifying set** *for f with images $f_i = \Phi_i(f)$, $i \in [\ell]$, if, for any pair of nonzero terms $\tau_1 = c_1 x^{e_1} \in f_i$ and $\tau_2 = c_2^{e_2} \in f_j$ with respective preimages $\sigma_1 \neq \sigma_2$, then there exists $k \in [m]$, such that of $\sigma_1(\alpha_k) \neq \sigma_2(\alpha_k)$.*

In other words, if $\alpha_1, \ldots, \alpha_m$ form a diversifying set for f with queries $f^{\bmod p_i}, i \in [\ell]$, then we can collect terms from the images $f^{\bmod p_i}$ according to their preimages. We assign to the degree-e term of $f^{\bmod p_i}$ the key $\boldsymbol{c}_e \in \mathsf{R}^m$, where $c_{e,j}$ is the coefficient of the degree-$(e \bmod p)$ term of $f(\alpha_j)^{\bmod p_i}$ for $j \in [m]$. We do this for every such term that gives a nonzero key. We then merely collect terms according to their keys.

A minor technicality is that a diversifying set does not need to distinguish a collision of terms from zero. Namely, we do not require that a particular preimage σ is such that $\sigma(\alpha_j) \neq 0$ for some $j \in [m]$. This is because we our algorithm does not need to use information coming from any non-singleton (formal) term τ that occurs in a query of f. If $cx^d \in f_i$ is a singleton term, then the degree-d term of f_{ij} is a singleton for $j \in [n]$. We thus need only consider exponents that appear in the support of f_i and each $f_{ij}, j \in [m]$.

Another way to describe a diversifying set is in terms of vector polynomials. We construct images. Namely, if we define the images of vector polynomials

$$\boldsymbol{f}(x) = (f(\alpha_1 x), f(\alpha_2 x), \ldots, f(\alpha_m(x))) = \sum_{i=1}^{t} \boldsymbol{c}_i \boldsymbol{x}^{e_i} \in \mathsf{R}^m[x], \tag{3.4}$$

$$\boldsymbol{f}_i(x) = \boldsymbol{f}(x)^{\bmod p_i} = (f(\alpha_1 x)^{\bmod p_i}, f(\alpha_2 x)^{\bmod p_i}, \ldots, f(\alpha_m x)^{\bmod p_i}), \quad i \in [\ell] \tag{3.5}$$

then $\alpha_1, \ldots, \alpha_m$ is a diversifying set for f with $f_1, \ldots, f_\ell$ if and only if any two nonzero terms from among the $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell$ sharing the same (vector) coefficient also share the same preimage. We probabilistically construct a diversifying set according to the following Lemma.

Lemma 3.5.4. *Let $f \in \mathsf{R}[x]$ be a polynomial of degree at most D and at most $T \geq 1$ nonzero terms, $f_1, \ldots, f_\ell$ images of f under sparsity-preserving maps $\Phi_1, \ldots, \Phi_\ell$. Furthermore, let $\mathcal{S} \subset \mathsf{R}$ be a finite regular-difference set with $\#\mathcal{S} \geq (1+\epsilon)D$, where $\epsilon > 0$ is a constant, and*

$$m = \left\lceil \frac{\log\left(\frac{1}{2}T^2(1+\ell/2)^2\mu^{-1}\right)}{\log \rho} \right\rceil. \tag{3.6}$$

Choose $\alpha_1, \ldots, \alpha_m$ independently and uniformly at random from $\mathcal{S}$. Then, with probability at least $1-\mu$, $\{\alpha_1, \ldots, \alpha_m\}$ forms a diversifying set for f with $f_1, \ldots, f_\ell$.

Proof. Consider a pair of nonzero preimages $\sigma_{i,u} \neq \sigma_{k,v} \in \mathsf{R}[x]$ as above. As $\sigma_{i,u} - \sigma_{k,v}$ has degree at most D, by the DLSZ Lemma (Thm. 2.5.7), for $\alpha \subset \mathcal{S}$ chosen uniformly at random, $(\sigma_{i,u} - \sigma_{k,v})(\alpha) = 0$ with probabilty at most $\frac{1}{\rho}$. Thus the probability that none of $\alpha_1, \ldots, \alpha_m$ distinguishes a fixed pair $\sigma_{i,u}$ and $\sigma_{k,v}$ is at most

$$\rho^{-m} \leq \frac{\mu}{\frac{1}{2}T^2(1+\ell/2)^2}. \tag{3.7}$$

Each nonzero preimage of a formal term in $f_1, \ldots, f_\ell$ is either one of the at most T singletons or the at most $\frac{1}{2}T\ell$ collisions of terms. It follows that there are in total fewer than $T(1+\frac{1}{2}\ell)$ distinct singletons and collisions of f that occur as a preimage in any of the f_i. From (3.7) we complete the proof taking the union bound over the at most $\frac{1}{2}T^2(1+\frac{1}{2}\ell)^2$ unordered pairs of distinct nonzero preimages. □

It is important to note that m (3.6) is logarithmic in T and ℓ. This implies that a multiplicative factor of m will not affect the soft-oh cost of our $\widetilde{\Omega}(\ell)$ interpolation algorithm. In the case where $\mathsf{R} = \mathbb{F}_q$ we have the following corollary.

Corollary 3.5.5. *Let q be a prime power, $u = \lceil \log_q(2D) \rceil$. Then Lemma 3.5.4 holds with $\mathcal{S} = \mathbb{F}_{q^u}$.*

Example 3.5.6. *Suppose* $f = 2x^{31} + 2^{33} + 7x^{110} + 7x^{161} \in \mathbb{F}_{23}[x]$. *Choose* $(p_1, p_2, p_3, p_4, p_5) = (5, 7, 11, 13, 17)$. *If we align terms from* $f_1, \ldots, f_5$ *in columns according to their preimage, we get*

$$\begin{array}{lrrrrrrrrrrrr} & 2x^{31} & & 2x^{33} & & 7x^{110} & & 7x^{161} & & \sigma_1 & & \sigma_2 & & \sigma_3 \\ \hline f_1 = f^{\bmod 5} = & & & 2x^3 & + & 7x^0 & + & & & 9x^1 & + & & & \\ f_2 = f^{\bmod 7} = & 2x^3 & + & & & & & 7x^0 & + & & & 9x^5 & & \\ f_3 = f^{\bmod 11} = & 2x^9 & + & & & & & 7x^7 & + & & & 9x^0 & & \\ f_4 = f^{\bmod 13} = & & & 2x^7 & + & 7x^6 & + & & & 9x^5 & + & & & \\ f_5 = f^{\bmod 17} = & 2x^{14} & + & 2x^{16} & + & & & & & & & & & 14x^8 \end{array} \tag{3.8}$$

where columns are labelled by the corresponding preimage, and

$$\sigma_1 = 2x^{31} + 7x^{110}, \quad \sigma_2 = 2x^{33} + 7x^{161}, \quad \sigma_3 = 7x^{110} + 7x^{161}.$$

Note, we cannot distinguish the preimages of $2x^{31}$ *with those of* $2x^{33}$. *Nor can we distinguish images of* $7x^{110}$ *with those of* $7x^{161}$ *or images of* σ_1 *with those of* σ_2.

However, if we choose $\alpha_1 = 22$, *and construct the images* $f_{i1} = f(\alpha_1 x) \bmod (x^p - 1)$, *we get*

$$\begin{array}{lrrrrrrrrrrrr} & 2x^{31} & & 2x^{33} & & 7x^{110} & & 7x^{161} & & \sigma_1 & & \sigma_2 & & \sigma_3 \\ \hline f_{11} = f(22x)^{\bmod 5} = & & & 21x^3 & + & 7x^0 & + & & & 14x^1 & + & & & \\ f_{21} = f(22x)^{\bmod 7} = & 21x^3 & + & & & & & 16x^0 & + & & & 5x^5 & & \\ f_{31} = f(22x)^{\bmod 11} = & 21x^9 & + & & & & & 16x^7 & + & & & 5x^0 & & \\ f_{41} = f(22x)^{\bmod 13} = & & & 21x^7 & + & 7x^6 & + & & & 14x^5 & + & & & \\ f_{51} = f(22x)^{\bmod 17} = & 21x^{14} & + & 21x^{16} & + & & & & & & & & & 0x^8 \end{array}$$

Now we can distinguish between the preimages of $7x^{110}$ *and* $7x^{161}$, *and those of* ψ_1 *and* ψ_2, *but not of* $2x^{31}$ *and* $2x^{33}$. *If we choose* $\alpha_2 = 9$, *we get*

$$\begin{array}{lrrrrrrrrrrrr} & 2x^{31} & & 2x^{33} & & 7x^{110} & & 7x^{161} & & \sigma_1 & & \sigma_2 & & \sigma_3 \\ \hline f_{12} = f(9x)^{\bmod 5} = & & & 2x^3 & + & 7x^0 & + & & & 9x^1 & + & & & \\ f_{22} = f(9x)^{\bmod 7} = & 4x^3 & + & & & & & 5x^0 & + & & & 9x^5 & & \\ f_{32} = f(9x)^{\bmod 11} = & 4x^9 & + & & & & & 5x^7 & + & & & 9x^0 & & \\ f_{42} = f(9x)^{\bmod 13} = & & & 2x^7 & + & 7x^6 & + & & & 9x^5 & + & & & \\ f_{52} = f(9x)^{\bmod 17} = & 4x^{14} & + & 2x^{16} & + & & & & & & & & & 12x^8 \end{array}$$

Now we know that the terms of f_1, f_4, *and* f_5 *of respective degrees* $3, 7$, *and* 16 *do not share the same preimages as the terms of of* f_2, f_3, *and* f_5 *of respective degrees* $3, 9$, *and* 14.

By inspection of the coefficients of the images f_{ij}, $i \in [2], j \in [5]$, *we can separate the terms of* $f_1, \ldots, f_5$ *into their respective columns as in* (3.8). *That is,* $\{22, 9\}$ *forms a diversifying set for* f *with* $f_1, \ldots, f_5$.

We should note that if, for a constructed set of images $\{f_{ij} : i \in [\ell], j \in [m]\}$, the degree-$d_1$ term of f_{1j} has the same coefficient as the degree-d_2 term of f_{2j} for all $j \in [m]$, that the terms may not necessarily have the same preimage, albeit with controlled probability.

3.5.3 Collecting images of terms

After we construct the images f_{ij}, we will build a dictionary that will allow us to collect terms from the images f_i according to their preimage. The images f_{ij}, $j \in [m]$, give the vectorized functions $\boldsymbol{f}_i$ (3.5), which we can write as

$$\boldsymbol{f}_i(x) = \sum_{k=1}^{t_i} \boldsymbol{c}_{ik} x^{e_{ik}},$$

where $t_i \leq t = \#f$, $\boldsymbol{c}_{ik} = (c_{ik1}, \ldots, c_{ikm}) \in \mathbb{F}_{q^s}^m$, and c_{ikj} is the coefficient of the degree-e_{ik} term of f_{ij}.

We use the vector coefficients $\boldsymbol{c}_{ik}$ in order to identify images of the same singletons or collisions of f. These coefficients will act as keys in a dictionary. The value associated with a key $\boldsymbol{c}$ will be a list of exponent-prime pairs (d, i) for every term $\boldsymbol{c}x^d$ occuring in an image $\boldsymbol{f}_i$, $i \in [\ell]$. Provided the probabilistic steps described in Sections 3.5.1 and 3.5.2 succeed, the keys whose value is a list of length at least $\ell/2$ correspond exactly to the nonzero terms of f.

After we have constructed the dictionary of terms, we iterate through its keys. For every key $\boldsymbol{c}$ that gives a set of at least $\ell/2$ congruences indexed by $\mathcal{I}_{\boldsymbol{c}} \subset [\ell]$,

$$\{e \equiv d_i \pmod{p} : i \in \mathcal{I}_{\boldsymbol{c}}\},$$

we construct an exponent $e \in [0..D]$ by way of Chinese remaindering on this system of congruences. This identifies every exponent with high probability. The coefficient c corresponding to an exponent e can be obtained from the degree-$(e \bmod p_i)$ term in f_i for some $i \in \mathcal{I}_{\boldsymbol{c}}$.

Theorem 3.5.7. *Algorithm 10 interpolates $f \in \mathbb{F}_q[x]$ with probability at least $2/3$. It admits a cost of $\widetilde{\mathcal{O}}(\log D \log T)$ $\mathbb{F}_{q^s}$-shifted queries of degree $\mathcal{O}(T \log D)$.*

Proof. We first show correctness. In each iteration of the outer while loop, the only probabilistic steps are the selection of the primes $p_1, \ldots, p_\ell$ and the $\alpha_1, \ldots, \alpha_m$. The call to procedure **GetPrimes**$(\lambda, n; 1/12)$ produces a set of either n primes or all the primes in $(\lambda, 2\lambda]$ with probability at least $11/12$. By Corollary 3.5.2, the primes $p_1, \ldots, p_\ell$ satisfy (3.5.i) with probability at least $11/12$. By setting $\lambda \geq \frac{10}{3} \ln D$, we have that $(\lambda..2\lambda]$ contains at least $2 \log_\lambda D$ primes per Corollary 2.4.5, such that, as $n \geq 2 \log_\lambda D$, any $n/2$ primes from $\mathcal{P}_{(\lambda..2\lambda]}$ will have a product exceeding D, such that $p_1, \ldots, p_\ell$ will satisfy (3.5.ii).

Algorithm 10: Majority-rule sparse interpolation of a straight-line program

Input: An extended black-box polynomial $f \in \mathbb{F}_q[x]$; $D \geq \deg(f)$; $T \geq \#f$.
Output: With probability at least $2/3$, a sparse representation of f

```
// Make random choices
```
$\lambda \leftarrow \lceil \max(21,\ \frac{5}{3}(T-1)\ln D,\ \frac{10}{3}\ln D) \rceil$;
if $\kappa \in (\lambda..2\lambda]$ **then** $\lambda \leftarrow 2\lambda$;
$n \leftarrow \lceil \max(8\ln(12T),\ 2\ln D \ln^{-1}\lambda) \rceil$;
$p_1, \ldots, p_\ell \leftarrow$ **GetPrimes**$(\lambda, n; 1/12)$;
$m \leftarrow \lceil \log\left(\frac{1}{2}T^2(1+\ell/2)^2\right) \rceil$;
$s \leftarrow \lceil \log_q D \rceil$;
Choose $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_{q^s}$ independently and uniformly at random;

```
// Construct images
```
for $i \leftarrow 1$ **to** ℓ **do**
  $f_i \leftarrow f \bmod p_i$;
  for $j \leftarrow 1$ **to** m **do** $f_{ij} \leftarrow f(\alpha_j x) \bmod p_i$;
  $\boldsymbol{f}_i = (f_{i1}, \ldots, f_{im}) \in \mathbb{F}_{q^s}^m[x]$;

```
// Collect terms from images and build sparse polynomial
```
$g \leftarrow 0 \in \mathbb{F}_{q^s}[x]$, a sparse polynomial;
for *every nonzero coefficient* $\boldsymbol{c}$ *appearing in a term* $\boldsymbol{c}x^{d_i}$ *in an image* $\boldsymbol{f}_i, i \in [\ell]$ **do**
  $\mathcal{I}_{\boldsymbol{c}} \leftarrow \{i \in [\ell] : \boldsymbol{f}_i$ has a term $\boldsymbol{c}x^{d_i}\}$;
  if $\#\mathcal{I}_{\boldsymbol{c}} < \ell/2$ **then continue**;
  $e \leftarrow$ solution in $[0..D]$ to congruences $\{e \equiv (d_i \bmod p_i) : i \in \mathcal{I}_{\boldsymbol{c}}\}$;
  $c \leftarrow$ coefficient of degree-e term of f_i for any $i \in \mathcal{I}_{\boldsymbol{c}}$;
  $g \leftarrow g + cx^e$;
return g;

By Lemma 3.5.4, $\{\alpha_1, \ldots, \alpha_m\}$ forms a diversifying set for f with $f_1, \ldots, f_\ell$ with probability at least $5/6$. By the union bound this guarantees the the algorithm succeeds with probability at least $2/3$.

We now consider the cost. The probe cost is clearly as stated. By Proposition 2.4.8, the cost of the call to **GetPrimes**$(\lambda, n; \frac{1}{12})$ costs $\widetilde{\mathcal{O}}(n \cdot \mathrm{polylog}(\lambda)) \subseteq \widetilde{\mathcal{O}}(n \cdot \mathrm{polylog}(T \log D))$ bit operations. The cost of constructing the at most T exponents via Chinese remaindering is, per Theorem 2.1.6, $\widetilde{\mathcal{O}}(T \log D)$ bit operations. These costs are dominated by the cost of the queries. □

Per the generalized DLSZ Lemma (Theorem 2.5.7), we can further generalize our algorithm

to any ring R containing a regular-difference set of cardinality at least $(1+\epsilon)D$ for some positive constant ϵ. This requires $\#\mathsf{R} \in \Omega(D)$, such that R-operations will cost $\Omega(\log D)$. Thus the number of bit operations required by Algorithm 10 will be at least $\widetilde{\Omega}(T \log^3 D)$. In case that f is given by an SLP $\mathbf{\Gamma}$, we have the following.

Corollary 3.5.8. *Algorithm 10 costs $\widetilde{\mathcal{O}}(LT \log^2 D)$ R-operations in the case that f is given by a length-L straight-line program. If, in addition, f is over $\mathbb{F}_q$, Algorithm 10 costs $\widetilde{\mathcal{O}}(LT \log^2 D(\log D + \log q))$ bit operations.*

3.6 Estimating degree and sparsity bounds on the interpolant

The interpolation algorithms presented in this chapter all require prior knowledge of bounds on the degree and sparsity of polynomial computed by the input straight-line program. This is a somewhat artifical constraint. In practise, one may need to efficiently derive such bounds.

For an n-variate polynomial f computed by a straight-line program $\mathbf{\Gamma}^f$ of length L, the total degree of f is at most 2^L; each instruction can double the total degree. We can construct tighter bounds, in time faster than the resulting sparse interpolation procedure should take. Recall that the straight-line program, encoded as a sequential list of instructions, gives a directed acyclic graph (representation) representation as an adjacency list, whereby we store the in-neighbours of the ith instruction. Variables and ring constants have no in-neighbours, and are represented as entries in the adjacency lists of their out-neighbours. This encoding is topologically sorted. A slightly better bound on $\deg(f)$ is 2^K, where K is the length of the longest path in the DAG.

A yet better method is to simply sequentially assign upper bounds $\mathcal{D}_j(\beta_i) \geq \deg_{x_j}(\beta_i)$ for all $j \in [n]$ and increasing i from 1 to L. The partial degree, with respect to some variable x, of an addition instruction will be the maximum of the bounds on those partial degrees of the inputs. For multiplication instructions it will be merely be the sum of the bounds on the inputs. Constructing a bound for the i-th instruction will be linear in the size of the resulting bound. The result β_i of the ith instruction Γ_i can have total degree at most 2^i, such that constructing these bounds for the ith instruction costs $\widetilde{\mathcal{O}}(i+n)$. Thus the integer arithmetic cost of such an approach over L instructions is thus at most $\widetilde{\mathcal{O}}(L^2 + Ln)$ bit operations. We summarize as follows:

Lemma 3.6.1. *Given a length-L SLP computing $f \in \mathsf{R}[x_1, \ldots, x_n]$, Algorithm 11 produces partial degree bounds $D_j \geq \deg_{x_j}(f)$ for $j \in [n]$. Its cost is $\widetilde{\mathcal{O}}(L^2 + Ln)$ bit operations.*

If we are not given *a priori* bounds on the sparsity of an extended black-box $f \in \mathbb{F}_q[x]$, we can construct shifted queries of f in order to probabilistically estimate its sparsity. Algorithm 12 gives pseudocode for our approach, which we describe and analyze here. We employ *repeated doubling*, i.e., we choose a guess $\widetilde{T}$ for $T \geq \#f$, test whether f is probably $\widetilde{T}$-sparse. If not, we

Algorithm 11: Determing partial degree bounds for an SLP

Input: $\mathbf{\Gamma}$, an SLP computing $f \in \mathsf{R}[x_1, \ldots, x_n]$, with L instructions of the form $\Gamma_i : \beta_i \leftarrow \gamma_{i1} \star \gamma_{i2}$.

Output: Bounds $D_j \geq \deg_{x_j}(f)$, for $j \in [n]$.

```
for i ← 1 to L do
    for k ∈ {1, 2} do
        if γ_ik ∈ R ∪ {x_1, ..., x_n} then
            for j ← 1 to n do D_j ← deg_{x_j}(γ_ik);
    if Γ_i is of the form β_i ← γ_i1 ± γ_i2 then
        for j ← 1 to n do D_j(β_i) ← max(D_j(γ_i1), D_j(γ_i2)));
    else if Γ_i is of the form β_i ← γ_i1 × γ_i2 then
        for j ← 1 to n do D_j(β_i) ← D_j(γ_i1) + D_j(γ_i2);
return {D_j = D_j(β_L) : j ∈ [n]};
```

double $\widetilde{T}$ and repeat. We choose $p \in \widetilde{\mathcal{O}}(\widetilde{T} \log D)$ to be a $1/2$-support prime for a fixed $2\widetilde{T}$-sparse polynomial g with degree probability at least $1 - \mu$.

Suppose that f has at least $2\widetilde{T}$ terms, and let $g \in \mathbb{F}_q[x]$ be comprised of the first $2\widetilde{T}$ terms of f. We further suppose that p is an $1/2$-support prime for g. In that case, g contains at least $\widetilde{T}$ singletons modulo $(x^p - 1)$ and either at least an additional singleton or collision of terms modulo $(x^p - 1)$, such that $\#(\mathrm{supp}(f) \bmod p) > \widetilde{T}$. By setting $s = \lceil \log_q(2D) \rceil$ and choosing $\alpha \in \mathbb{F}_{q^s}$ at random, we guarantee for a fixed formal term $\tau = cx^e \in f^{\bmod p}$, with a nonzero preimage, that $e \in \mathrm{supp}(f(\alpha x)^{\bmod p})$, with probability at least $1/2$ by the DLSZ Lemma. Thus if we set $m = \lceil \log(p/\mu) \rceil$ and choose $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_{q^s}$ independently and uniformly at random, we guarantee with probability $1 - \mu$ that for any formal term $cx^e \in f^{\bmod p}$ with a nonzero preimage there exists $j \in [m]$ such that $f(\alpha_j x)^{\bmod p}$ has a nonzero term of degree e, regardless of f.

In summation, if these probabilistic steps succeed, then if f has at least $2\widetilde{T}$ terms, then the vector function

$$(f^{\bmod p})(\boldsymbol{\alpha} x) \stackrel{\text{def}}{=} (f(\alpha_1 x)^{\bmod p}, \ldots, f(\alpha_m x)^{\bmod p}) \in \mathbb{F}_{q^s}[x],$$

has $\#f(\boldsymbol{\alpha} x)^{\bmod p} > \widetilde{T}$. Thus in this case we double $\widetilde{T}$ and repeat this approach, otherwise we probabilistically guarantee $\#f(\boldsymbol{\alpha} x) \leq T = 2\widetilde{T}$. Given that $t = \#f \leq D + 1$, this will entail at most $\log(D + 1) + 1$ iterations. Thus, if each iteration succeeds with probability $1/\mu$, where

Algorithm 12: Sparsity Estimation of an SLP over $\mathbb{F}_q$

Input: An extended black-box polynomial $f \neq 0 \in \mathbb{F}_q$; $D \geq \deg(f)$.

Output: With probability at least $2/3$, returns $\widetilde{T} \geq \#\mathrm{supp}$.

$\widetilde{T} \leftarrow 1$;
$\mu \leftarrow \lfloor \log(D+1) + 1 \rfloor$;
$s \leftarrow \lceil \log_q D \rceil$;
while `true` **do**
  $\lambda \leftarrow \lceil \max(21, \frac{160}{3}\widetilde{T} \ln D/\mu) \rceil$;
  $p \leftarrow$ **GetPrime**$(\lambda; \mu/3)$;
  $m \leftarrow \log(3p/\mu)$;
  Choose $\boldsymbol{\alpha} \in \mathbb{F}_{q^s}^m$ uniformly at random;
  if $\#f(\boldsymbol{\alpha}\boldsymbol{x})^{\bmod p} > \widetilde{T}$ **then return** $T = 2\widetilde{T}$;
  else $\widetilde{T} \leftarrow 2\widetilde{T}$;

$\mu = 3\lfloor \log(D+1) + 1 \rfloor$, the algorithm will succeed with probability at least $1/3$ Moreover, an iteration will necessarily terminate if $\widetilde{T} \geq \#f$, such that we never output $T \geq 2t$.

Each iteration comprises three probabilistic parts: the success of the subroutine call to **GetPrime**; the event that p is a $1/2$-support prime; the selection of $\alpha_1, \ldots, \alpha_m$. Per techiques similarly seen throughout the chapter, we leave it to the reader to verify that each succeeds with probability at least $\mu/3$.

The cost in the ith iteration is dominated by the query cost. Each iteration entails m $\mathbb{F}_{q^s}$-shifted queries of degree $\widetilde{\mathcal{O}}(\widetilde{T} \log D/\mu)$. Per the assumptions of Remark 3.2.4, this admits a $\mathbb{F}_{q^s}$ operation cost of $\widetilde{\mathcal{O}}(mT \log D) \subset \widetilde{\mathcal{O}}(T \log D)$, as m is polylogarithmic in $T \log D$. Accounting for all $\mathcal{O}(\log D)$ iterations, as well as the cost of finite field arithmetic, gives a total bit cost of $\widetilde{\mathcal{O}}(T \log^2 D(\log D + \log q))$.

Theorem 3.6.2. *Algorithm 12 takes an extended black box polynomial $f \neq 0 \in \mathbb{F}_q[x]$ and $D \geq \deg(f)$, and with probability at least $2/3$, produces $T \in [t..2t]$, where $t = \#f$. Otherwise the algorithm produces $T < t$.*

Its cost is $\widetilde{\mathcal{O}}(T \log D)$ $\mathbb{F}_{q^s}$-shifted queries, where $s = \lceil \log_q(2D) \rceil$.

This cost equals that of majority-rule interpolation. In particular, when f is given by an SLP, we have the following:

Corollary 3.6.3. *For $f \in \mathbb{F}_q[x]$ given by a length-L SLP, Algorithm 12 admits a bit operation cost of $\widetilde{\mathcal{O}}(LT \log^2 D(\log D + \log q))$.*

3.7 Conclusions and open questions

In this section we gave two algorithms, iterative and majority-rule interpolation, for interpolating a T-sparse extended black-box polynomial with degree at most D. Both have cost softly linear in the sparsity bound T, and cubic in $\log D$, with a $\log D$ factor being absorbed by the cost of ring arithmetic in the case of majority-rule interpolation.

The algorithms are both *small primes* interpolation algorithms, in that one reconstructs the exponents via Chinese remaindering on a set of primes polylogarithmic in D. They both use a set of $\widetilde{\mathcal{O}}(\log D \log T)$ modular images belonging to a R-vector space of cardinality as large as $\widetilde{\Omega}(T \log^2 D)$, i.e. either $\mathsf{R}[x]^{\bmod pq}$ or $\mathbb{F}_{q^s}[x]^{\bmod p}$, where $p \in \widetilde{\Omega}(T \log D)$, $q \in \widetilde{\Omega}(\log D)$, and $s = \lceil \log_q(2D) \rceil$. It appears we need a set of this size in order to accomplish two tasks: uniquely map *most* of the terms of f (or a difference $f - f_{\text{sparse}}$ for a sparse polynomial f_{sparse}) to terms of significantly smaller degree; and map terms in a manner that we can identify most or all resulting terms in images according to their preimage. In iterative interpolation, we accomplish both of these tasks by introducing a factor p into the degree of every query for a $5/8$-support prime p. In majority-rule interpolation, we accomplish the former by choosing a set of small primes as exponent moduli, and the latter via generalized divesification.

3.7.1 Extensions to other models

All the algorithms here extend to *Laurent polynomials*. E.g., given an extended black-box polynomial $f \in \mathsf{R}[x^{\pm 1}]$ with $\operatorname{absdeg}(f) \leq D$, we can interpolate the polynomial $x^D f \in \mathsf{R}[x]$ with degree at most $2D$, from which we can recover f with no additional polynomial cost factor. That is,

Proposition 3.7.1. *The algorithms of Chapter 3 extend to Laurent polynomials f with $\operatorname{absdeg}(f) \leq D$, at the same costs as is stated. In particular, Algorithms 6–8, 10–11, and procedure **IterativeSI** may be adapted to interpolate $f \in \mathsf{R}[x^{\pm 1}]$, and Algorithms 9, 10, and 12 can be adapted to $f \in \mathbb{F}_q[x^{\pm 1}]$.*

As the interpolation algorithms of the chapter construct exponents by way of Chinese remaindering, we could also merely find solutions to congruences in the range $[-D..D]$, for sufficiently many primes p.

The algorithms herein also may be adapted to *algebraic black boxes* (Defn. 1.4.2). Recall that an algebraic black box for $f \in \mathsf{Z}[x^{\pm 1}]$ is a slightly weaker version of an extended black box, allowing for evaluation of f over any algebraic extension of Z. Here, if $p = \operatorname{char}(\mathsf{Z})$ is the characteristic of Z, then we cannot query f to yield $f^{\bmod mp}$ for $m \in \mathbb{Z}_{>0}$, as $(x^{pm} - 1)$ factors into $(x^m - 1)^p$. For the purposes of our algorithms, it then suffices that we merely choose query degrees not divisible by p. E.g., if our algorithm were to select primes from $(\lambda..2\lambda] \ni p$, we merely

double λ. If our algorithm were to construct the first ℓ primes, we instead have it construct the first 2ℓ primes.

Proposition 3.7.2. *The algorithms of Chapter 3 may be adapted to $f \in \mathsf{Z}[x^{\pm 1}]$ (or $f \in \mathbb{F}_q[x^{\pm 1}]$ for those as stated in Prop. 3.7.1) given by an algebraic black box, with no additional cost.*

Another problem to explore is whether small primes interpolation translates well to black-box numerical models, e.g., in the case that the coefficient ring is $\mathbb{C}$. In this setting, a black box may serve as an extended black box. One can construct $f(\alpha x) \bmod p$ given by an algebraic or extended black box for $f \in \mathbb{C}[x]$ by evaluating it at the points $\alpha \cdot \exp(2\pi i j/p)$, $j \in [p]$, and interpolating the result via a prime-length FFT, e.g., Rader's FFT [Rad68] or Bluestein's FFT [Blu70]. In order to obtain a robust numerical interpolation analog to **IterativeSI** or Algorithm 10, one would need a careful analysis of the numerical stability of such an approach.

Giesbrecht and Roche give a numerical version of their diversified interpolation algorithm in [GR11]. In order to collect terms according to preimage via coefficients, one requires that the coefficients of $f(\alpha x)$ are more than machine precision from each other. They achieve this by choosing α to be a random root of unity for a fixed choice of p. They also require that all the coefficients of f each are sufficiently far from zero. In the case of generalized diversity, we would need to construct a set of complex α, such that any pair of colliding sums of terms, i.e., preimages, disagree for some α. This problem reduces to finding a set $\mathcal{V} \subset \mathbb{C}$ such that for an unknown set $\mathcal{P}$ of unknown T-sparse polynomials of degree at most D, for every $\sigma \in \mathcal{P}$ there exists $\alpha \in \mathcal{V}$ such that $\sigma(\alpha)$ is bounded distance away from zero. This is the numerical version of Zippel's *zero avoidance problem*.

3.7.2 Open problems

We mention some open problems we have not deeply explored. First, we ask whether one can break the soft-oh $T \log^3 D$ barrier, i.e.:

Problem 3.7.3. *Given a length-L SLP $\mathbf{\Gamma}^f$ computing $f \in \mathsf{R}[x]$ with bounds $D \geq \deg(f)$ and $T \geq \#f$, can we interpolate f with a bit and ring operation cost that is $\widetilde{o}(LT \log^3 D)$.*

A related problem is to construct lower bounds for the cost of this interpolation problem. How to solve this problem is unclear to us. As our techniques heavily rely on the construction of modular images of f, a good first problem to solve would be to show that for any set of polynomial moduli $m_1, \ldots, m_s \in \mathsf{R}[x]$, there exists a sparse polynomial f that is zero modulo some or all of those images. This problem is closely tied to the sparse multiple problem: given a polynomial $m(x) \in \mathsf{R}[y]$ and bounds D and T, does there exist T-sparse $f \in \mathsf{R}[x]$ that is a multiple of $m(x)$? This is an interesting and nontrivial problem in its own right.

Another related problem is whether we can do sparse polynomial identity testing in subquadratic time, i.e.:

Problem 3.7.4. *Given a length-L SLP $\mathbf{\Gamma}^f$ computing $f \in \mathsf{R}[x]$, with bounds $D \geq \deg(f)$ and $T \geq \#f$, can we test if $f = 0$ with a bit and ring operation cost that is $\widetilde{o}(LT^2 \log^2 D)$?*

If we could reduce this cost to linear in T, the Monte Carlo interpolation algorithms we presented could be made Las Vegas.

Chapter 4

Output-sensitive algorithms for sumset and sparse polynomial multiplication

4.1 Introduction

In this chapter we use some of the techniques from univariate sparse interpolation in order to obtain an asymptotically fast Monte Carlo algorithm for the multiplication of sparse univariate polynomials $f, g \in \mathbb{Z}[x]$ with integer coefficients. The algorithm is joint work with Daniel S. Roche and appears in [AR15]. A challenge of this problem is that the number of terms, and the size of the coefficients may vary greatly. For example,

$$(x^8 + x^7 + x^5 + x^4 + x^3 + x + 1)(x^7 + x^6 + x^5 + x^2 + x + 1) =$$
$$x^{15} + 2x^{14} + 2x^{13} + 2x^{12} + 2x^{11} + 4x^{10} + 4x^9 + 4x^8 + 4x^7 + 4x^6 + 4x^5 + 2x^4 + 2x^3 + 2x^2 + 2x + 1,$$

whereas

$$(x^8 - x^7 + x^5 - x^4 + x^3 - x + 1)(x^7 + x^6 + x^5 - x^2 - x - 1) = x^{15} - 1. \tag{4.1}$$

We present a multiplication algorithm that is sensitive to the number of terms in the output. As a subroutine, we give a Monte Carlo algorithm for computing the **sumset** (also known as the *Minkowski sum*) of two finite sets of integer vectors $\mathcal{A}, \mathcal{B} \subset \mathbb{Z}^n$, defined by

$$\mathcal{A} \oplus \mathcal{B} \stackrel{\text{def}}{=} \{\boldsymbol{a} + \boldsymbol{b} : \boldsymbol{a} \in \mathcal{A}, \boldsymbol{b} \in \mathcal{B}\},$$

where $\boldsymbol{a} + \boldsymbol{b}$ is the componentwise sum. The runtime of our multiplication algorithm depends on the cardinality of the sumset of the supports of the inputs. We define the **possible support** as

$$\operatorname{poss}(f, g) \stackrel{\text{def}}{=} \operatorname{supp}(f) \oplus \operatorname{supp}(g),$$

and remark that this contains $\mathrm{supp}(fg)$ as a subset. Throughout this chapter, we will let $h = fg$, and use the following bounds.

$$\begin{aligned} C &= \max\left(\|f\|_\infty, \|g\|_\infty\right), & D &= \deg(fg), \\ S &= \#\mathrm{poss}(f,g), & T &= \#f + \#g + \#h, \\ T_{\mathrm{in}} &= \#f + \#g, & T_h &= \#h. \end{aligned}$$

We note that given inputs f and g, the parameters S and T are not immediate. As we saw in the example above they may vary for inputs with fixed degrees and a fixed number of terms. We state our results.

Theorem 4.1.1. *Let $\mathcal{A}, \mathcal{B} \subset [0..D]$ be finite sets and let $S = \#(\mathcal{A} \oplus \mathcal{B})$. Then **Sumset**$(\mathcal{A}, \mathcal{B})$ computes $\mathcal{A} \oplus \mathcal{B}$ with probability at least $2/3$ and a bit operation cost of $\widetilde{\mathcal{O}}(S \log D)$.*

For the purposes of sparse polynomial multiplication, we will take $\mathcal{A} = \mathrm{supp}(f)$ and $\mathcal{B} = \mathrm{supp}(g)$.

Theorem 4.1.2. ***SparseMultiply**(f, g), computes a sparse representation of fg, with probability at least $2/3$ and a bit operation cost of*

$$\widetilde{\mathcal{O}}(S \log D + T \log C).$$

Both **Sumset** and **SparseMultiply** trivially extend to the n-dimensional case via *Kronecker substitution*. For those cases we replace the $\log D$ factors appearing in Theorems 4.1.1 and 4.1.2 with $n \log D$.

We also give an analysis of the Las Vegas sparse multiplication algorithm of Cole and Hariharan and show that their algorithm can be easily modified to compute a product with cost $\widetilde{\mathcal{O}}(S(\log C + \log D))$.

4.1.1 Context

As we have mentioned, the number of terms in the product fg may vary. We state some rudimentary bounds for the size of a sumset and the sparsity of a product.

Lemma 4.1.3. *Let $\mathcal{A}, \mathcal{B} \subset \mathbb{Z}$ be finite, and $f, g \in \mathbb{Z}[x]$ be nonzero polynomials, with at least one having more than one term. Then the following hold.*

$$\begin{aligned} \#\mathcal{A} + \#\mathcal{B} - 1 \;\leq\; \#(\mathcal{A} \oplus \mathcal{B}) \;&\leq\; (\#\mathcal{A})(\#\mathcal{B}), \\ 2 \;\leq\; \#fg \;&\leq\; (\#f)(\#g). \end{aligned} \tag{4.2}$$

From this Lemma, we have that

$$\widetilde{\mathcal{O}}(T_{\text{in}}) \subseteq \widetilde{\mathcal{O}}(S) \subseteq \widetilde{\mathcal{O}}(T_{\text{in}}^2), \qquad \widetilde{\mathcal{O}}(T_h) \subseteq \widetilde{\mathcal{O}}(T_{\text{in}}^2). \tag{4.3}$$

In an extreme case, we may have that $(\#f)(\#g) \in \Omega(T^2)$, e.g., if $\#h \leq \#f + \#g$.

A naive lower bound for the runtime of any algorithm for sparse multiplication or sumset is softly linear in the respective **problem size**: the combined bit size of the problem's inputs and outputs. If we suppose that f, g, and $h = fg$ are all encoded by their sparse representations and satisfying:

(4.1.i) the same number of bits are used to represent every nonzero coefficient; and

(4.1.ii) the same number of bits are used to represent every partial exponent;

then the total bit size of the inputs and outputs for the multiplication problem is

$$\widetilde{\mathcal{O}}(T(\log D + \log C)). \tag{4.4}$$

This follows from the fact that the coefficients of h are less than TC^2, and so the $\mathcal{O}(T)$ coefficients of h require at most $\mathcal{O}(T \log(TC^2)) \subset \widetilde{\mathcal{O}}(T \log C)$ bits.

As the $S \log D$ term in the cost stated in Theorem 4.1.2 has hidden $\operatorname{polylog}(\log C)$ factors, the runtime of the multiplication algorithm exceeds the problem size by an additive factor $\widetilde{\mathcal{O}}((S - T) \log D \cdot \operatorname{polylog}(\log C))$.

Similarly, if we suppose that the same number of bits is used to represent every component of every vector $\boldsymbol{v}$ in $\mathcal{A}, \mathcal{B}$, or $\mathcal{A} \oplus \mathcal{B}$, then the combined bit size of the inputs and outputs to the sumset problem is $\widetilde{\mathcal{O}}(S \log D)$. In that sense we achieve an optimal algorithm.

There may be instances where, if we relax the assumptions (4.1.i) and (4.1.ii), we may achieve smaller problem sizes. We list a few such cases:

- If most terms of f, g, and h have relatively small coefficients, and the remaining terms have coefficients that, in comparison, are exponential in magnitude; or similarly we could have that a small proportion of terms have degree that is exponential in comparison to that of the rest.
- Multivariate cases where the exponents of $h \in \mathbb{Z}[\boldsymbol{x}]$ may be better served using *sparse exponents*. Consider the case such that $f \in \mathbb{Z}[x_1, \ldots, x_n]$ is the sum of monomials of total degree 3, and $h = f^2$. Then h would be comprised of all monomials of total degree 6. In our sparse representation we would represent the term $x_1^3 x_2^3$ of h as $(1, (3, 3, 0, 0, \ldots, 0))$, where the exponent has length-n. In this case perhaps a more natural encoding would be $(1, ((3, 1), (3, 2)))$.

- If the height of the product is considerably smaller than the height of the inputs. For instance, if we let $\Phi_n(x) \in \mathbb{Z}[x]$ denote the nth **cyclotomic polynomial**, i.e., the minimal polynomial of the complex primitive nth roots of unity, then Φ_n divides $(x^n - 1)$. However, Erdös proved that the heights of Φ_n can grow arbitrarily large [Erd46]. In particular, he showed $\|\Phi_n\|_\infty$ is not bounded by a polynomial in n.

We do not consider further potential speedup in these "unbalanced" instances.

We further preface our results by noting that for an overwhelming majority of conceivable inputs, either a dense multiplication algorithm or the *grade-school method* of sparse polynomial multiplication is softly optimal. If the number of terms in h is asymptotically quadratic in the number of terms of f and g, then the grade-school method will compute the product in softly linear time, under the assumption (4.1.i). There are two ways by which the product h can have subquadratic terms. First, we can have the case that the possible support $\mathrm{poss}(f, g) = \mathrm{supp}(f) \oplus \mathrm{supp}(g)$ has cardinality subquadratic in terms of $\#\mathrm{supp}(f) + \#\mathrm{supp}(g)$. The easiest way to construct such an example is if we take f and g to be dense, for which a fast dense multiplication algorithm is softly linear. Second, we can have **coefficient cancellation**, whereby terms with a common exponent sum together to produce zero coefficients. This is the case in the carefully-constructed example (4.1). For that example we took $f = \Phi_{15}(x)$, and $g = (x^{15} - 1)/f$.

In "most" cases, there will be little to no coefficient cancellation. To substantiate this, fix a pair of support sets $\mathcal{A}, \mathcal{B} \subset \mathbb{Z}^n_{\geq 0}$, and consider the product

$$\left(\sum_{\boldsymbol{a} \in \mathcal{A}} c_{\boldsymbol{a}} \boldsymbol{x}^{\boldsymbol{a}}\right) \left(\sum_{\boldsymbol{b} \in \mathcal{B}} c_{\boldsymbol{b}} \boldsymbol{x}^{\boldsymbol{b}}\right),$$

where we treat the coefficients $c_{\boldsymbol{a}}$ and $c_{\boldsymbol{b}}$ as indeterminates. This product has support that is a subset of $\mathcal{A} \oplus \mathcal{B}$ regardless of the choice of coefficients. A coefficient cancellation occurs at exponent $\boldsymbol{e} \in \mathcal{A} \oplus \mathcal{B}$ if and only if the $c_{\boldsymbol{a}}$ and $c_{\boldsymbol{b}}$ are a solution to the multilinear diophantine equation

$$\Phi_{\boldsymbol{e}} \stackrel{\text{def}}{=} \sum_{\substack{(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{A} \times \mathcal{B} \\ \boldsymbol{a} + \boldsymbol{b} = \boldsymbol{e}}} c_{\boldsymbol{a}} c_{\boldsymbol{b}} = 0.$$

The set of such solutions will be within a hyperplane of $\mathbb{Z}^{\mathcal{A} \otimes \mathcal{B}}$. Thus, for any $c_{\boldsymbol{v}}$ appearing as a variable in $\Phi_{\boldsymbol{e}}$, and fixing the choices of $c_{\boldsymbol{u}}$, $\boldsymbol{u} \in \mathcal{A} \oplus \mathcal{B} \setminus \{c_{\boldsymbol{v}}\}$, there will be at most one choice of $c_{\boldsymbol{v}} \in \mathbb{Z}$ that gives a coefficient cancellation. Moreover, it is not necessarily the case that an asympotically significant number of coefficient cancellations will simultaneously occur.

That said, we can construct structured examples whereby the algorithm herein may be advantageous over either dense or grade-school sparse multiplication. One such example is taking high powers of a sparse polynomial.

Example 4.1.4. *Suppose $f = x^7+3x^5y^2+y^7$. Then, for instance, f^8 has 42 terms and partial degrees at most 56, whereas f^{16} has 98 terms and partial degrees at most 112. Were we to compute f^{16} as $(f^8)^2$ using a dense multiplication algorithm, the resulting dense representation of f^{16} would require storage for $(112+1)^2 = 12769 \gg 98$ coefficients. Note, moreover, that $(\#(f^8))^2 = 1764 \gg 98$ terms, the number of terms in the product is significantly subquadratic.*

This example gives a product with subquadratically many terms because the sumset of the support of f^8 with itself is structured in a way that there will exist many pairs of exponents in $\operatorname{supp}(f^4)$ whose componentwise sums agree.

If f is t-sparse, then f^n may have as many as $\binom{t+n-1}{n}$ distinct terms. If $n \gg t$, then $\#(f^n) \ll \#(f^{\lceil n/2 \rceil})\#(f^{\lfloor n/2 \rfloor})$

$$\binom{t+\lceil n/2 \rceil - 1}{\lceil n/2 \rceil}\binom{t+\lfloor n/2 \rfloor - 1}{\lfloor n/2 \rfloor} \gg \binom{t+n-1}{n},$$

so that, the number of terms in the product $f^n = f^{\lceil n/2 \rceil} f^{\lfloor n/2 \rfloor}$ may be significantly subquadratic. Moreover, if f is *sparse* and n is not too large, we may expect that f^n is not significantly dense, such that it does not lend itself to dense arithmetic.

Lastly, a softly-linear sparse multiplication algorithm would be a theoretically meaningful result, and would provide a sparse analogue to dense polynomial multiplication, which can be made softly-linear under a dense encoding satisfying (4.1.i) via FFT-based methods. The algorithm herein is a meaningful step towards that goal.

4.2 Previous sparse multiplication algorithms

Table 4.1 gives a comparison of univariate multiplication algorithms. The simplest way to multiply sparse polynomials is by the **grade-school method**: multiply every term of f with every term of g. Using dense multiplication over the integers modulo a prime $q > 2C+1$ gives a cost of $\widetilde{\mathcal{O}}(D \log C)$. To compute a product of terms $c_1\boldsymbol{x}^{e_1} \cdot c_2\boldsymbol{x}^{e_2} = c_1c_2\boldsymbol{x}^{e_1+e_2}$, we merely need to take the product of the coefficients and the sums of the exponents. Both of these operations have softly linear cost. In the worst case we may have $(\#f)(\#g) \in \Omega(T^2)$, such that the grade school method has a worst-case cost of $\widetilde{\mathcal{O}}(T^2 \log D)$. We describe a few more sophisticated methods hereafter.

4.2.1 Multiplication via interpolation

One way to compute the product fg is to merely interpolate the product. Given sparse representations of $f, g \in \mathsf{R}[x]$, we realize a black box for fg by simply evaluating $(fg)(\alpha)$ as $f(\alpha)g(\alpha)$. A single term $\tau = cx^e$ can be evaluated at α in $\mathcal{O}(\log D)$ ring evaluations by way of the square and multiply technique. This gives the following lemma.

Algorithm	Reference	Section	Soft-oh cost	Type[1]
dense	Ch. 8, [VG03]		$D \log C$	Det
grade school		4.2	$T^2(\log C + \log D)$	Det
BaseCaseMultiply[2]	[AGR13]	4.2.2.3	$T(\log C + \log D)\log^2 D$	MC
Prony (Alg. 14)[2]		4.2.2	$T(\log C + \log D^2)$	MC
Cole–Hariharan[3]	[CH02]	4.2.3	$S(\log C + \log D)$	LV
SparseMultiply	[AR15]	4	$S \log D + T \log C$	MC
Prony (Alg. 13)[3]		4.2.2.1	$\widetilde{\mathcal{O}}(S \log D + T(\log C + \log^2 D)$	MC
ProductSparsity		4.4	$\widetilde{\mathcal{O}}(T_{\text{in}} \log C + S \log D)$	MC

1. MC = Monte Carlo; LV = Las Vegas; Det = Deterministic.
2. **BaseCaseMultiply** requires a bound $T_h \geq \#(fg)$.
3. Additional $\widetilde{\mathcal{O}}(\log^3 D \cdot \text{polylog}(T \log C))$ precomputation cost to find a large prime $p \in \widetilde{\mathcal{O}}(D + T \log C)$.
4. Additional precomputation cost to find a prime q for which $\mathbb{Z}_q$ contains a sth root of unity ω, where $s > D$ is *smooth*.

Table 4.1: A comparison of sparse multiplication and product sparsity estimation

Lemma 4.2.1. *Given sparse representations of $f, g \in \mathsf{R}[x]$ and $\alpha \in \mathsf{R}$, we can compute the evaluation $(fg)(\alpha)$ in $\mathcal{O}(T_{\text{in}} \log D)$ ring operations.*

We will also use fast multipoint evaluation in Section 4.4, to evaluate fg more efficiently over a geometric sequence, with fg embedded in an appropriate ring. We do this for the purposes of efficiently estimating the sparsity of the product.

More generally, we can perform fast multipoint evaluation of a sparse polynomial $f = \sum_{i=1}^{t} c_i x^{e_i}$ over a field K by way of the transposed Vandermonde product

$$\begin{bmatrix} \omega^{0e_1} & \omega^{0e_2} & \cdots & \omega^{0e_t} \\ \omega^{1e_1} & \omega^{1e_2} & \cdots & \omega^{1e_t} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(t-1)e_1} & \omega^{(t-1)e_2} & \cdots & \omega^{(t-1)e_t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} f(\omega^0) \\ f(\omega^1) \\ \vdots \\ f(\omega^{t-1}) \end{bmatrix}.$$

By Theorem 2.1.10 and the *Transposition Principle* (Thm. 2.1.11), we can compute this system in $\widetilde{\mathcal{O}}(t)$ field operations. We trivially can generalize this to produce $s > t$ evaluations at consecutive powers of ω by treating f as an s-sparse polynomial with coefficients $c_i = 0$ for $i > t$. Thus we have the following Lemma.

Lemma 4.2.2. *Given sparse representations of $f, g \in \mathsf{K}[x]$, and $\omega \in \mathsf{K}$, we can compute the evaluations $h(\omega^i)$ where $h = fg, s \geq \max(\#f, \#g)$, and for $i \in [0..s)$, in $\widetilde{\mathcal{O}}(s)$ field operations.*

4.2.2 Multiplication via "large primes" interpolation

One way to compute the product h is by way of Prony's algorithm. Such an algorithm would entail two parts: probabilistically estimating $T_h = \#h$, and then interpolating h using an appropriate root of unity ω of multiplicative order exceeding D.

4.2.2.1 Estimating the sparsity of $h = fg$

We use Algorithm 5 in order to estimate the sparsity of the product. We can use $(\#f)(\#g) \in \mathcal{O}(T_{\text{in}}^2)$ as the initial naive upper bound on $\#h$ we provide the algorithm. As the algorithm cost is at least softly linear in $\widetilde{\mathcal{O}}(T_{\text{in}})$, we input an initial guess of $T_{\min} = T_{\text{in}}$ for the sparsity of h. We will need to embed fg in $\mathbb{Z}_p[x]$ for a prime p. The algorithm requires that $p \geq 3DT_{\text{in}}^4$. We can find such a prime probabilistically by way of the Miller–Rabin primality test with a cost of $\widetilde{\mathcal{O}}(\log^3 D + \log^3 T_{\text{in}}) = \widetilde{\mathcal{O}}(\log^3 D)$ (Lemma 2.4.10). Reducing f and g modulo p potentially costs $\widetilde{\mathcal{O}}(T_{\text{in}} \log C)$.

We note that this approach would produce an estimate for $\#(h \bmod p)$ and not necessarily f. As the product h has at most T_{in}^2 terms with height at most $T_{\text{in}}C^2$, the number of primes for which a term of f disappears is at most $\widetilde{\mathcal{O}}(T_{\text{in}}^2 \log C)$, such that by looking for primes $p \in \widetilde{\mathcal{O}}(D + T_{\text{in}} \log C)$, we can guarantee that none of the coefficients of h dissappear for a choice of p with high, constant probability (e.g. $99/100$).

In order to find a root of unity $\omega \in \mathbb{Z}_p$ of multiplicative order exceeding D, we can discover a primitive element ω via the algorithm of Lemma 2.4.11 with cost $\widetilde{\mathcal{O}}(\log^3 p)$. We can alternatively instead run Algorithm 5 for $\mathcal{O}(\log \log p)$ choices of $\omega \in \mathbb{Z}_p^*$, such that with high constant probability, at least one of the choices of ω is a primitive element.

The cost of multipoint evaluation for each instance we run Algorithm 5 is then $\widetilde{\mathcal{O}}(T_{\text{in}} + T_h)$ operations modulo p per Lemma 4.2.2. Algorithm 13 describes this approach, albeit without explicit parameters. We summarize with the following Lemma:

Lemma 4.2.3 (Algorithm 13)**.** *There exists an algorithm which, given sparse representations of $f, g \in \mathbb{Z}[x]$, produces $\widetilde{T_h} \in [\max(T_h + 1, T_{\text{in}})..\max(2T_h, T_{\text{in}})]$ with probability at least $2/3$, and produces $\widetilde{T_h} \leq T_h$ otherwise. Its cost is $\widetilde{\mathcal{O}}(T_{\text{in}} \log C + T \log D)$ and an additional $\widetilde{\mathcal{O}}(\log^3 D \cdot \operatorname{polylog}(T_{\text{in}} \log C))$ bit operations of precomputation.*

In Section 4.4, we improve on this approach to avoid the cost of having to construct a large prime p.

Algorithm 13: Prony sparsity estimation for a sparse polynomial product

Input: Sparse polynomials $f, g \in \mathbb{Z}[x]$.
Output: With high probability, $\widetilde{T_h} \geq \max(T_{\text{in}}, T_h)$.
Parameters: $C = \max(\|f\|_1, \|g\|_1)$; $D = \deg(fg)$; $T_{\text{in}} = \#f + \#g$.
Precomputation: Choose a prime $p \in \widetilde{\Theta}(D + T_{\text{in}} \log C)$ via Miller–Rabin

1 **repeat** $\mathcal{O}(\log \log p)$ **times**
2 Execute Algorithm 4 on $h \bmod p$, with bounds $T_{\text{in}}^2 \geq T_h$, $D \geq \deg(h)$, and initial guess $T_{\min} = T_{\text{in}}$, using fast multipoint evaluation (Lemma 4.2.2) in order to produce evaluations of h;
3 **return** maximum output produced by Algorithm 4;

4.2.2.2 Interpolating the product via Prony's algorithm

Once we have an estimate $\widetilde{T_h}$, where $T_{\text{in}} \leq \widetilde{T_h} \leq \max(2T_h, T_{\text{in}})$, we can use Prony's algorithm to interpolate h; however, we need to work in a setting where we can do discrete logarithms efficiently. Namely, we choose a prime q such that $q-1$ is divisible by a power of two s exceeding D. Then $\mathbb{Z}_q$ contains an sth root of unity ω_s for which discrete logarithms in base ω_s are inexpensive.

Namely, for $\alpha = \omega_s^e$, the least significant bit of e is 0 if and only if $\alpha^{s/2} \bmod q = 1$. Then, if we determine this bit to have value e_0, we have that $(\alpha/\omega_s^{e_0}) = (\omega_s^2)^{e'}$, the ith least significant bit of $e' \in \mathbb{Z}$ is the $(i+1)$th least significant bit of e for $i \geq 1$. We can then learn the least significant bit of e' in the same manner, continuing until we have all the bits of e. In this way we can perform logarithms with cost $\widetilde{\mathcal{O}}(\log q)$ operations modulo q, or $\widetilde{\mathcal{O}}(\log^2 q)$ bit operations. We can generalize this technique for q such that $q-1$ has a large *smooth* divisor $s > D$.

If we choose $q \in \widetilde{\Omega}(D + \widetilde{T_h} \log C)$, then we can guarantee with high constant probability that none of the coefficients of h disappear modulo q, such that $\operatorname{supp}(h \bmod q) = \operatorname{supp}(h)$. As we will show in Section 4.5, the coefficients can then be obtained with cost $\widetilde{\mathcal{O}}(T(\log C + \log D))$, via **SparseProductCoeffs**, which combines ideas from Prony's method and Chapter 3. Thus the total cost becomes $\widetilde{\mathcal{O}}(T(\log C + \log^2 D))$.

We do not have a tight analysis of the precomputation of q and ω_s. The machinery developed in Section 2.4 to find primes q for which $\mathbb{Z}_q$ contains a pth root of unity for a prime p within a specified range. This is because there are significantly fewer smooth integers than primes. Given q, we can discover such a value ω_s via the approach of Lemma 2.4.11, with a cost of $\widetilde{\mathcal{O}}(\log^2 q)$ bit operations.

Lemma 4.2.4 (Algorithm 14)**.** *There exists an algorithm that, takes sparse representations of f and g and precomputed values of:*

Algorithm 14: Sparse polynomial multiplication via Prony

Input: Sparse polynomials $f, g \in \mathbb{Z}[x]$.
Output: With high probability, $h = fg$.
Parameters: $C = \max(\|f\|_1, \|g\|_1)$; $D = \deg(fg)$; $T_{\text{in}} = \#f + \#g$.
Precomputation: Find a prime $q \in \widetilde{\mathcal{O}}(D + T_{\text{in}} \log C)$ such that $s = 2^{\lceil \log D + 1 \rceil}$ divides $q - 1$, and an sth root of unity ω.

1 $\widetilde{T_h} \leftarrow$ Output of Alg. 13 with inputs f, g, T_{in}, D;
2 Interpolate $h \bmod q$ via Prony's algorithm (Alg. 3), using root of unity ω;
3 **return** **SparseProductCoeffs**$(f, g, \text{supp}(h \bmod q); {}^{99}/_{100})$

- *a prime q for which $s|q-1$ for some smooth integer $s > D$, and*
- *an sth primitive root of unity ω,*

and produces $h = fg$ with probability ${}^2/_3$ and cost $\widetilde{\mathcal{O}}(T(\log C + \log^2 D))$.

4.2.2.3 Multiplication via "small primes" interpolation

We can also use the interpolation methods of Chapter 3, by constructing an extended black box for h. In order to compute a p-query of $fg \in \mathsf{R}[x]$, we reduce f and g modulo $(x^p - 1)$ and take their product in $\mathsf{R}[x]^{\bmod p}$. Per Lemma 3.2.3, this gives us the following costs.

Lemma 4.2.5. *Given sparse representation of $f, g \in \mathsf{R}[x]$ and an integer $p \in \mathbb{Z}_{>0}$, one can compute the following with given costs:*

- *A p-query of fg in $\widetilde{\mathcal{O}}(T_{\text{in}} + p)$ ring operations and an additional $\widetilde{\mathcal{O}}(T_{\text{in}} \log D)$ bit operations.*
- *A shifted p-query of fg in $\widetilde{\mathcal{O}}(T_{\text{in}} \log D + p)$ ring operations.*

We can thus, for instance, choose a prime q, construct images $(f \bmod q), (g \bmod q) \in \mathbb{Z}_q[x]$, and use iterative two-prime interpolation (Procedure **IterativeSI**) to interpolate $fg \bmod q$.

Our multiplication procedure **BaseCaseMultiply** supplies the extended black box polynomial realized by **ProductExtBB** to **IterativeSI**. From Theorem 3.4.8 and Lemma 4.2.5, this gives the following cost:

Lemma 4.2.6. *Given sparse representations of f and $g \in \mathsf{R}[x]$ and bounds $D \geq \deg(fg)$ and $T \geq \#f$, one can compute the sparse representation of $h = fg \in \mathsf{R}[x]$ with cost $\widetilde{\mathcal{O}}(T_{\text{in}} \log D + T \log^3 D)$ R-operations via Procedure **IterativeSI**.*

We use **BaseCaseMultiply** as a subroutine in our main multiplication algorithm.

Procedure ProductExtBB(f, g, p, α)

Input: Sparse polynomials $f, g \in \mathsf{R}[x]$; $p \in \mathbb{Z}_{>0}$; $\alpha \in \mathsf{R}$.
Output: Returns the (shifted) p-query $h(x) \bmod p$ ($h(\alpha x) \bmod p, resp.$), where $h = fg$.

1 $f_{p,\alpha} \leftarrow f(\alpha x) \bmod p$;
2 $g_{p,\alpha} \leftarrow g(\alpha x) \bmod p$;
3 **return** $(f_{p,\alpha} g_{p,\alpha}) \bmod p$, computed via dense arithmetic;

Procedure BaseCaseMultiply(f, g, D, T)

Input: Sparse polynomials $f, g \in \mathsf{R}[x]; D \geq \deg(fg); T \geq \#(fg)$.
Output: Returns a sparse representation of $h = fg$ with probability at least $2/3$.

1 Let $h_\blacksquare$ be an extended black box for $h = fg$ given by **ProductExtBB**$(f, g, \cdot, \cdot)$;
2 **return** **IterativeSI**$(h_\blacksquare, D, T; 2/3)$;

4.2.3 The Cole–Hariharin Las Vegas multiplication algorithm

In [CH02], Cole and Hariharin give a Las Vegas algorithm for sparse convolution. Their multiplication algorithm is used as a subroutine in sparse pattern matching problems. Their algorithms for sparse pattern matching problems are in turn used as subroutines in sparse convolution. We describe their approach and give an analysis of their method.

Fix a finite alphabet $\mathcal{A}$, e.g. a finite set, containing a **wildcard character** "$*$". Generally we will assume $\mathcal{A} \subset \mathbb{Z}_{\geq 0}$ with $* = 0$. Cole and Hariharin considered the problem of finding all instances of a **pattern** $\boldsymbol{f} = (f_0, \ldots, f_{m-1}) \in \mathcal{A}^m$ appearing in a **text** $\boldsymbol{g} = (g_0, \ldots, g_{n-1}) \in \mathcal{A}^n$. We identify $\boldsymbol{f}$ with $f = \sum_{i=0}^{m-1} f_i x^i$ and $\boldsymbol{g}$ with $g = \sum_{i=0}^{n-1} g_i x^i$. The non-wildcard characters of $\boldsymbol{f}$ and $\boldsymbol{g}$ correspond to the nonzero terms of f and g.

4.2.3.1 Wildcard matching problem

Without loss of generality we can assume $n \leq 2m$. For longer texts $\boldsymbol{g}$ we can break $\boldsymbol{g}$ into contiguous segments of length $2m$ that begin at every mth index, and then solve the wildcard matching problem for the $\mathcal{O}(n/m)$ segments. We suppose $\Sigma = [0..C]$.

We say the pattern $\boldsymbol{f}$ **occurs at position** i in $\boldsymbol{g}$ if, for all $j \in [0..m)$, either $f_j = g_{i+j}$ or one of f_j and g_{i+j} is a wildcard symbol. The **wildcard matching problem** is to find all instances $i \in [0..n-m)$ for which $\boldsymbol{p}$ occurs at position i in $\boldsymbol{t}$.

Their solution is to construct a pair of products. The first constructs a new pattern $\boldsymbol{f}'$ and text $\boldsymbol{g}'$ defined by $f'_j = 0$ if $f_j = *$, and $f'_j = 1$ otherwise, and similarly for $\boldsymbol{g}'$. Then if we let

$f' = \sum_{j=0}^{m-1} f'_j x^j, g' = \sum_{j=0}^{n-1} g'_j x^j$, and $h' = (f'g') \bmod n = \sum_{j=0}^{n-1} h' x^i \in \mathbb{Q}[x]$, the coefficient of the x^i term of h' gives the number of non-wildcard characters that align when f_0 is aligned with g_i.

They then construct $\boldsymbol{f}'' \in \mathbb{Q}^{2m}$ and text $\boldsymbol{g}'' \in \mathbb{Q}^{2n}$, where

$$(f''_{2j}, f''_{2j+1}) = \begin{cases} (0,0) & \text{if } f_j = *, \\ (f_j, 1/f_j) & \text{otherwise,} \end{cases} \quad \text{for } j \in [0..m)$$

$$g''_j = \begin{cases} (0,0) & \text{if } g_j = *, \\ (1/g_j, g_j) & \text{otherwise.} \end{cases} \quad \text{for } j \in [0..n).$$

Let $f'' = \sum_{j=0}^{2m-1} f''_j x^j$, $g'' = \sum_{j=0}^{2n-1} g''_j x^j$ and $h'' = f''g'' \bmod (x^{2n} - 1) = \sum_{j=0}^{n-1} h''_i x^i \in \mathbb{Q}[x]$. Then, if f occurs at position i in g, then $h''_i = 2h'$; otherwise $|h'' - 2h'| \geq 1/C$. By computing the reduced product h'' to $\mathcal{O}(\log C)$ bits of precision, we can detect all positions where $\boldsymbol{f}$ occurs in $\boldsymbol{g}$. By (Thm. 1, [CH02]), the cost of computing such a product is $\widetilde{\mathcal{O}}(n \log C)$. One could also embed $\boldsymbol{f}''$ and $\boldsymbol{g}''$ over $\mathbb{Z}_q$, for a prime $q > C$, instead of over $\mathbb{Q}$; this would require an additional precomputation cost to obtain q.

4.2.3.2 Shifted wildcard matching problem

The **shift matching problem** is the problem of finding all indices i for which $\boldsymbol{f}$ occurs at some **shift** of $\boldsymbol{g}$. That is, to find all $i \in [0..n-m)$ for which there exists a constant k_i such that, for all $j \in [0..m)$, either $f_j = g_{i+j} + k_i$ or one of f_j or $g_{i+j} = 0$. We again assume an alphabet $\mathcal{A} = [0..C]$. Cole and Hariharan give a $\widetilde{\mathcal{O}}(n \log C)$ algorithm for this problem as well.

Their algorithm again first computes h' as in the wildcard matching problem, to determine the number of non-wildcards that align for each alignment of $\boldsymbol{p}$ starting at the ith index of text $\boldsymbol{t}$. They then choose an Cth complex primitive root of unity ω, and construct the complex-valued pattern $\boldsymbol{p}''$ given by $p''_j = \omega^{p_j}$ for $j \in [0..m)$, and $\boldsymbol{t}''$ similarly. Then, letting $h'' = (\sum_{j=0}^{m-1} p''_j x^j)(\sum_{j=0}^{n-1} t''_j x^j) \bmod (x^n - 1)$, we have that $\boldsymbol{p}$ occurs at some shift of $\boldsymbol{t}$ is and only if the x^i coefficient of h'' is ω^{k_i} times the x^i coefficient of h'. Again by performing a dense product numerically, this gives a $\widetilde{\mathcal{O}}(n \log C)$ algorithm.

We can also work in non-numerical settings, e.g., any ring containing a root of unity of multiplicative order exceeding C. We could work, for instance, over $\mathbb{Z}_\ell^2$, where $\ell \geq C$ is an integer. In that setting we have the ℓth root of unity $\omega = \ell + 1$, satisfying $\omega^e \bmod \ell^2 = e\ell + 1$, such that logarithms with base ω are easy to compute. We will use fast logarithms over $\mathbb{Z}_{\ell^2}$ in Section 4.3 in order to quickly determine the exponents of a particular product.

4.2.3.3 Fast sparse convolution

Now we consider the product of $f, g \in \mathbb{N}[x]$, where $m = \deg(f)$, $n = \deg(g)$, and $D = m + n$. In this setting there is no coefficient cancellation, such that $T_h = S$. In order to multiply $f, g \in \mathbb{Z}[x]$, we would have to write f as $f^+ - f^-$ and g as $g^+ - g^-$, where each of f^+, f^-, g^+, g^- have strictly nonnegative coefficients, and then compute h as

$$h = f^+g^- - f^+g^- - f^-g^+ + f^-g^-.$$

They use a *repeated doubling* approach, whereby their algorithm makes a guess $\widetilde{S} = 1$ for S and continues doubling its guess until they verify that the product is computed correctly. We consider an iteration where we have $\widetilde{S} > S$, such that f, g, and h are all $\widetilde{S}$-sparse.

Their algorithm fixes a prime $p \in (2D..4D]$ and an integer r, and for various choices of $a \in \mathbb{Z}_p^*$, constructs products of the form

$$h_a = (f(x^a) \bmod p \, g(x^a) \bmod p) \bmod r \in \mathbb{N}[x], \tag{4.5}$$

for various choices of $a \in \mathbb{Z}_p^*$. Note that, for every exponent $e \in \mathrm{supp}(h)$, h_a has at least one or possibly both terms of degree $ae \bmod r$ and $ae + p \bmod r$. In this setting say a pair of distinct exponents $e_1, e_2 \in \mathrm{supp}(\phi)$, where $\phi \in \mathbb{N}[x]$ **collide** for a choice of a if any pair of the potential resulting exponents agree. That is, if

$$(ae_1 \bmod p) \bmod r \in \{(ae_2 \bmod p) + kp \bmod r : k \in \{0, 1, 2\}\}.$$

We extend the notion of a **singleton** in this setting to mean a non-colliding term of h, As $ae \bmod p$ is uniformly distributed over $\mathbb{Z}_p^*$ for $e \neq 0$ and a chosen uniformly at random from $\mathbb{Z}_p^*$, we have that the probability that e and e^* collide for a choice of a is less than $8/r$. Thus, if r is chosen to be at least $16\widetilde{S}$, the probability that e and e^* collide for a choice of a is at most $\widetilde{S}/2$.

By the union bound, the probability that $e \in \mathrm{supp}(h)$ collides with any other exponent of h for a choice of a is at most $1/2$ (assuming correctness of the bound $\widetilde{S}$). Similarly, an exponent of f also avoids collision with other exponents of f in the image f_a with probability less than $1/2$, and similarly for g. By choosing $m = \lceil \log(9\widetilde{S}) \rceil$ $a \in \mathbb{Z}_p^*$, then with probability at least $2/3$, every term of f, g, and h avoids collision in f_a, g_a, and h_a for some choice of a, with probability at least $2/3$.

Now fix a choice of a and define

$$f_a = \left(f(x^a) \bmod p\right) \bmod r = \sum_{i=0}^{r-1} f_{ai} x^i, \qquad g_a = \left(g(x^a) \bmod p\right) \bmod r = \sum_{i=0}^{r-1} g_{ai} x^i.$$

Such that $h_a = (f_a g_a)^{\bmod r}$. We construct a pattern $\boldsymbol{f}' \in \mathbb{Z}_4^r$ and text $\boldsymbol{g}' \in \mathbb{Z}_4^{2r}$ given by

$$f'_{-i \bmod r} = \begin{cases} 0 & \text{if } f_{ai} = 0 \\ 1 & \text{if } f_{ai}x^i \text{ is a singleton} \\ 3 & \text{if } f_{ai}x^i \text{ is a multiple.} \end{cases} \qquad \text{for } i \in [n],$$

$$g'_i, g'_{i+r} = \begin{cases} 0 & \text{if } g_{ai} = 0 \\ 1 & \text{if } g_{ai}x^i \text{ is a singleton} \\ 2 & \text{if } g_{ai}x^i \text{ is a multiple.} \end{cases} \qquad \text{for } i \in [n],$$

Then, if the pattern $\boldsymbol{f}'$ occurs at locations $i, i+p \in [0..r)$ in $\boldsymbol{g}'$, then the the x^i coefficient of h_a is a sum of products of pairs of coefficients of singleton terms of f_a and g_a. Let I' be the set of such indices i.

We can then use the shift matching problem to determine, for each $i \in I'$, whether the degree-i term of h_a is a singleton. We construct a text $\boldsymbol{f}''$ and pattern $\boldsymbol{g}''$ defined by

$$f''_j = \begin{cases} -e \bmod D & \text{if } f_{aj}x^j \in f_a \text{ has a preimage } cx^e \in f \text{ for some } c \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

$$g''_j = \begin{cases} e & \text{if } g_{aj}x^j \in g_a \text{ has a preimage } cx^e \in g \text{ for some } c \neq 0; \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } j \in [r].$$

In other words, $f''_j = 0$ only if $f_{aj}x^j$ is not a singleton, and similarly for g''_j. If $\boldsymbol{f}''$ occurs at some shift of $\boldsymbol{g}''$ at an index $i \in \mathcal{I}'$, we have that $f''_j + g''_{i+j} = k_i$ for all $j \in [0..m)$ and some constant k_i. In other words, the degree-i term of h_a is the image of a single term of h of degree k_i. For each $i \in \mathcal{I}'$, such that $\boldsymbol{f}''$ occurs at a shift of $\boldsymbol{g}''$ we note the shift k_i corresponding to the degree of the preimage of the degree-i term of h_a. We constuct the corresponding coefficients of h by way of computing h_a over a ring $\mathbb{Z}_u$, where $u > 2T_{\text{in}}^4 C^2 \geq 2\|h\|_\infty$, and taking the resulting coefficients over the range $(\lfloor -u/2 \rfloor .. \lfloor u/2 \rfloor]$. We maintain a sum $\widetilde{h}$ of these constructed terms.

The resulting cost is $\widetilde{\mathcal{O}}(T_{\text{in}} \log D)$ to produce $f_a, \boldsymbol{f}', \boldsymbol{f}'', g_a, \boldsymbol{g}'$, and $\boldsymbol{g}''$, $\widetilde{\mathcal{O}}(\widetilde{S})$ for the wildcard-matching problem, $\widetilde{\mathcal{O}}(\widetilde{S} \log D)$ for the shift-matching problem, and $\widetilde{\mathcal{O}}(\widetilde{S} \log C)$ to compute h_a. Accounting for all $\mathcal{O}(\log S)$ values of a does not affect the soft-oh cost. With probability at least $2/3$, if $\widetilde{S} > S$, we produce every term of h. Moreover, as we know every constructed term is a true term of h, and the coefficients of f, g, and h are all positive, we know we have $\widetilde{h} = h$ when $f(1)g(1) = \widetilde{h}(1)$.

Thus the expected cost of this approach is $\widetilde{\mathcal{O}}(S(\log C + \log D))$. We remark that Cole and Hariharan stated a cost of $\widetilde{\mathcal{O}}(S \log^2 m)$. This is because they used $\mathcal{O}(\log m)$ choices of a as opposed to $\mathcal{O}(\log \widetilde{S})$. We should remark that it was not their aim to minimize the cost with respect to $\log m$. We summarize our analysis as a Theorem.

Theorem 4.2.7 (Cole–Hariharan algorithm)**.** *There exists an algorithm which, given $f, g \in \mathbb{Z}[x]$, computes $h = fg$ with an expected cost of $\widetilde{\mathcal{O}}(S(\log C + \log D))$ bit operations, and an initial $\widetilde{\mathcal{O}}(\log^3 D)$ bit operations to obtain a prime $p > D$.*

As future work, we would like to remove this precomputation cost by choosing an appropriate prime power in place of p in (4.5).

4.3 A softly-linear Monte Carlo algorithm for sumset

Here we present a Monte Carlo algorithm for sumset that first appeared in [AR15]. Throughout Section 4.3, we let $\mathcal{A}, \mathcal{B} \in [0..D]$, $\mathcal{S} = \mathcal{A} \oplus \mathcal{B}$, $R = \#\mathcal{A} + \#\mathcal{B}$, and $S = \#\mathcal{S}$. We will prove that our sumset procedure **Sumset**$(\mathcal{A}, \mathcal{B}, D)$ produces $\mathcal{A} \oplus \mathcal{B}$ with probability $2/3$ and a bit cost of $\widetilde{\mathcal{O}}(S \log D)$. We note that one could use the Las Vegas Cole–Hariharan algorithm to obtain the same complexity.

We will compute the sumset of $\mathcal{A}, \mathcal{B} \subset \mathbb{Z}^n$ as $\mathrm{supp}(FG)$, where

$$F = \sum_{\boldsymbol{a} \in \mathcal{A}} \boldsymbol{x}^{\boldsymbol{a}}, \quad G = \sum_{\boldsymbol{b} \in \mathcal{B}} \boldsymbol{x}^{\boldsymbol{b}}.$$

This polynomial product is special in that all of the coefficients in the inputs are positive, from which it follows that no coefficient cancellation will occur in the product FG.

Our algorithm comprises two signficant parts. First, we estimate $S = \#\mathrm{supp}(F, G)$. Once we have an estimate for S, we use it to interpolate the product FG with degree at most $2D$ and satisfying

$$R - 1 \le \#\mathrm{supp}(FG) = S \le R^2.$$

4.3.1 Estimating the cardinality of sumset

In this section we give a Monte Carlo for estimating S, the cardinality of $\mathcal{A} \oplus \mathcal{B}$, i.e., $\#\mathrm{supp}(FG)$. This algorithm probabilistically produces $\widetilde{S}$ such that

$$S \le \widetilde{S} < 8S.$$

If the algorithm fails, it will either produce an error message `fail`, or a value $\widetilde{S} < S$.

Using Lemma 3.3.6 and the bound $\#H \le R^2$, we first probabilistically select a good prime $p \in \widetilde{\mathcal{O}}(R^4 \log D)$, such that $\#\mathrm{supp}(H^{\bmod p})$ determines $\#\mathrm{supp}(H)$. Define

$$F_p \stackrel{\text{def}}{=} F^{\bmod p}, \quad G_p \stackrel{\text{def}}{=} G^{\bmod p}, \quad H_p \stackrel{\text{def}}{=} F_p G_p \in \mathbb{Z}_p[x].$$

Then $\deg(H_p) < 2p$ and each nonzero term cx^e of H corresponds to a nonzero term of H_p of degrees $e \bmod p$ and $(e \bmod p) + p$. Thus,

$$\begin{aligned} \#H &\le \#H_p, && \text{if } p \text{ is a good prime for } H, \text{ and} \\ \#H_p &\le 2\#H && \text{for any choice of } p. \end{aligned}$$

We will make an initial guess of $\widetilde{S} = R$ for S, and double $\widetilde{S}$ until we are confident that we have a value of $\widetilde{S} > S$. Given our guess $\widetilde{S}$, we probabilistically select a $1/2$-support prime $q \in \widetilde{\mathcal{O}}(\widetilde{S} \log p)$ for a $\widetilde{S}$-sparse polynomial with degree less than $2p$, and compute $H_{pq} \stackrel{\text{def}}{=} H_p^{\bmod q}$. Note that

$$\begin{aligned} \#H_p/2 < \#H_{pq}, &\qquad \text{if } q \text{ is a } 1/2\text{-support prime for } H_p, \text{ and} \\ \#H_{pq} \leq \#H_p &\qquad \text{for any choice of } q. \end{aligned}$$

We test whether H_{pq} has fewer than $\widetilde{S}/2$ nonzero terms. If the test accepts we produce $\widetilde{S}$, otherwise we double the value of $\widetilde{S}$ and repeat. If $\widetilde{S} > 4S$, then we have $\#H_{pq} \leq 2S < \widetilde{S}/2$, and hence H_{pq} is $\widetilde{S}/2$-sparse regardless of the choice of p and q. Thus the test accepts and we produce $\widetilde{S}$. Conversely if p and q are chosen correctly, then $\#H_{pq} > S/2$, and so $S < \widetilde{S}$ if the test fails. Thus, provided the good prime p and $1/2$-support primes q are chosen correctly, when the test terminates we will have $\widetilde{S}$ such that $S \leq \widetilde{S} \leq 8S$. If p or any of the q are not chosen correctly, then we can erroneously produce $\widetilde{S} < S$.

In order to compute H_{pq}, we embed the problem in $\mathbb{Z}_{R^2}$ and use fast dense arithmetic. As the total length of H is less than R^2, any coefficient of $H_{pq} \in \mathbb{Z}[x]$ will be in the range $[0..R^2)$. Procedure **SumsetSize** describes the approach.

We remind the reader that, by our convention, if a procedure used as a subroutine, e.g. GetPrime in **SumsetSize**, produces `fail`, then its calling procedure is understood to produce `fail` as well. We prove the following:

Proposition 4.3.1. ***SumsetSize***$(\mathcal{A}, \mathcal{B}, D)$ *produces* $\widetilde{S} \in [S..8S)$*, where* $S = \#(\mathcal{A} \oplus \mathcal{B}) \leq \widetilde{S}$ *with probability at least* $2/3$*. If the procedure fails it either produces* $\widetilde{S} < S$*, or it produces* `fail`*. It admits a cost of* $\widetilde{\mathcal{O}}(R \log D + S)$ *bit operations.*

Proof. We first analyze the correctness of the procedure. By Lemma 3.3.6, a prime chosen from $(\lambda..2\lambda]$ is a good prime for R^2-sparse $H_0 \in \mathbb{Z}[x]_D$ with probability at least $11/12$, and **GetPrime** produces a prime $p \in (\lambda..2\lambda]$ with probability at least $11/12$. Whether or not p is a good prime, we have that $\#H_p \leq 2S$.

By Lemma 3.4.3, a prime q chosen from $(\lambda_q..2\lambda_q]$ is a $1/2$-support support prime with probability at least $1 - \mu/12$, and **GetPrime** selects $q \in \mathcal{P}_{(\lambda_q..2\lambda_q]}$, also with probability at least $1 - \mu/12$.

The while loop will terminate before $\widetilde{S}$ reaches a value at least $8S < 8R^2$. Thus the number of iterations of the while loop will be less than $\log(8R^2) - \log(R) + 1 = \mu^{-1}$. It follows that the algorithm succeeds with probability at least $2/3$.

The algorithm can only fail if a call to **GetPrime** fails, in which case **SumsetSize** produces `fail`, or **SumsetSize** produces $\widetilde{S} < S$.

Procedure SumsetSize($\mathcal{A}, \mathcal{B}$)

Input: $\mathcal{A}, \mathcal{B} \subset [0..D]$, where $D \in \mathbb{Z}_{>0}$
Output: With probability at least $2/3$, $\widetilde{S} \in \mathbb{Z}$ such that $\widetilde{S}/4 < S \leq \widetilde{S}$. Otherwise produces $\widetilde{S} < S$ or fails.

$R \leftarrow \#F + \#G$;
$F, G \leftarrow \sum_{a \in \mathcal{A}} x^a, \sum_{b \in \mathcal{B}} x^b \in \mathbb{Z}_{R^2}[x]$;

$\lambda \leftarrow \max(21, \lceil 20R^2(R^2-1)\log D \rceil)$;
$p \leftarrow$ **GetPrime**$(\lambda; 1/12)$;
$(F_p, G_p) \leftarrow (F \text{ rem } (x^p - 1), G \text{ rem } (x^p - 1))$;

$\widetilde{S} \leftarrow R$;
$\mu \leftarrow 1/(4 + \log R)$;
while `true` **do**
  $\lambda_q \leftarrow \max(21, \lceil \frac{120}{3}(\widetilde{S} - 1)\mu^{-1} \log(2p)) \rceil)$;
  $q \leftarrow$ **GetPrime**$(\lambda_q; \mu/12)$;
  $(F_{pq}, G_{pq}) \leftarrow \big(F_p \bmod (R^2, x^q - 1), G_p \bmod (R^2, x^q - 1)\big)$;
  $H_{pq} \leftarrow F_{pq}G_{pq} \bmod (R^2, x^q - 1)$, computed via dense arithmetic;
  if $2\#H_{pq} \leq \widetilde{S}$ **then return** $\widetilde{S}$;
  $\widetilde{S} \leftarrow 2\widetilde{S}$;

The cost of producing the images F_p, G_p is, per Lemma 3.2.3, $\widetilde{\mathcal{O}}(R \log D)$ operations in $\mathbb{Z}_{R^2}$, or a bit cost of $\widetilde{\mathcal{O}}(R \log D)$. The cost of producing the prime p is $\text{polylog}(p) \in \widetilde{\mathcal{O}}(R \log D)$, per Lemma 2.4.7.

The cost of producing F_{pq} and G_{pq} for a single iteration of the while loop similarly is $\widetilde{\mathcal{O}}(R \log p)$ operations, and the cost of producing q costs $\widetilde{\mathcal{O}}(\text{polylog}(q) \log(\mu^{-1})) = \widetilde{\mathcal{O}}(\text{polylog}(q) \log(R))$. The cost of computing the product $F_{pq}G_{pq}$ and reducing the result modulo $(x^q - 1)$ is $\widetilde{\mathcal{O}}(q)$ operations in $\mathbb{Z}_{R^2}$. As $q \in \widetilde{\mathcal{O}}(\sim \log p) \subset \widetilde{\mathcal{O}}(S \log \log D)$. Thus, accounting for all $\widetilde{\mathcal{O}}(\log R)$ iterations of the while loop, the procedure costs

$$\widetilde{\mathcal{O}}(R \log D + S \log R \log \log D) \subseteq \widetilde{\mathcal{O}}(R \log D + S),$$

completing the proof. □

4.3.2 Computing sumset

Once we are armed with the bound $\widetilde{S} \geq S = \#\mathrm{supp}(FG)$, we probabilistically construct the product $H = FG$. Our approach is to construct a pair of images

$$H_1 = F_1 G_1 \bmod (\ell^2, x^p - 1), \quad H_2 = F(\omega_\ell x) G(\omega_\ell x) \bmod (\ell^2, x^p - 1),$$

where p is a good prime for H, $\ell = 4D \geq \max(\deg(H), \|H\|_1)$, and $\omega_\ell = \ell + 1$. Note that

$$\omega_\ell^e = (\ell + 1)^e \equiv 1 + e\ell \pmod{\ell^2},$$

that is, ω_ℓ is an ℓth primitive root of unity modulo ℓ^2, and we can easily recover e from ω_ℓ^e. In other words, discrete logarithms with base $\omega_\ell \in \mathbb{Z}_{\ell^2}$ are easy to compute.

A single term $\tau = cx^e \in H$ maps to $\tau_1 = cx^{e \bmod p}$ in H_1 and $\tau_2 = c\omega_\ell^e x^{e \bmod p}$ in H_2. If cx^e avoids collision modulo $(x^p - 1)$ we can recover ω_ℓ^e by computing the quotient of the coefficients of the degree-$(e \bmod p)$ terms τ_2 and τ_1.

Proof of Theorem 4.1.1. The probabilistic steps of **Sumset** are the four calls to subroutines, each of which succeed with probability at least $1/15$, and the random selection of p. By Lemma 3.3.6, p is a good prime for H with probability at least $1/15$, provided the estimate $\widetilde{S} \geq \#H$ given by **SumsetSize** is correct. By the union bound **Sumset** succeeds with probability at least $2/3s$.

By Proposition 4.3.1, $\widetilde{S} \in \widetilde{\mathcal{O}}(S)$. The calls to **BaseCaseMultiply** thus each cost $\widetilde{\mathcal{O}}(S \log^3 p)$ ring operations in $\mathbb{Z}_{\ell^2}$, of a bit cost of $\widetilde{\mathcal{O}}(S \log^3 p \log \ell) \subseteq \widetilde{\mathcal{O}}(S \log D)$ bit operations. The cost of constructing a sumset element for each nonzero term of $c_z x^e$ of H_1 costs $\widetilde{\mathcal{O}}(S)$ arithmetic operations on integers $c_1, c_2 \in [0..\ell^2)$, or $\widetilde{\mathcal{O}}(S \log D)$ bit operations. These dominate the cost of **Sumset**. □

In the case that $\mathcal{A}, \mathcal{B} \in [-D..D]$ instead, we use the same techniques. We merely construct the exponents of FG modulo $\ell > 2D$, and then translate the exponents from the range $[0..2D]$ to $[-D..D]$. We can extend to the multidimensional setting where $\mathcal{A}, \mathcal{B} \in [-D..D]^n$ by instead computing $\mathcal{A} \oplus \mathcal{B}$ as $\mathrm{supp}(FG)$, where now

$$F = \sum_{\boldsymbol{a} \in \mathcal{A}} \boldsymbol{x}^{\boldsymbol{a}}, \quad G = \sum_{\boldsymbol{b} \in \mathcal{B}} \boldsymbol{x}^{\boldsymbol{b}} \in \mathbb{N}[\boldsymbol{x}^{\pm 1}],$$

and using Kronecker substitution . This gives us the following corollary:

Corollary 4.3.2. *There exists an algorithm that, given $\mathcal{A}, \mathcal{B} \subset [-D..D]^n$, produces $\mathcal{A} \oplus \mathcal{B}$ with probability at least $2/3$ and cost $\widetilde{\mathcal{O}}(Sn \log D)$.*

Procedure Sumset($\mathcal{A}, \mathcal{B}, D$)

Input: $\mathcal{A}, \mathcal{B} \subset [0..D]$; $D \in \mathbb{Z}_{>0}$.
Output: With probability at least $2/3$, $\mathcal{A} \oplus \mathcal{B}$.

$F, G \leftarrow \left(\sum_{a\in\mathcal{A}} x^a, \sum_{b\in\mathcal{B}} x^b\right) \in \mathbb{Z}_{R^2}[x]$;
$\widetilde{S} \leftarrow$ **SumsetSize**$(\mathcal{A}, \mathcal{B}, D; 1/15)$;

$\lambda \leftarrow \max(21, \lceil 25T(T-1)\log D\rceil)$;
$p \leftarrow$ **GetPrime**$(\lambda; 1/15)$;
$\ell \leftarrow 2D+1$;
$(F_1, G_1) \leftarrow \left(\sum_{a\in\mathcal{A}} x^{a \bmod p}, \sum_{b\in\mathcal{B}} x^{b \bmod p}\right) \in \mathbb{Z}_{\ell^2}[x]^2$;
$(F_2, G_2) \leftarrow \left(\sum_{a\in\mathcal{A}} (a\ell+1)x^{a \bmod p}, \sum_{b\in\mathcal{B}} (b\ell+1)x^{b \bmod p}\right) \in \mathbb{Z}_{\ell^2}[x]^2$;
$H_1 \leftarrow$ **BaseCaseMultiply**$(F_1, G_1, \widetilde{S}, 2p; 1/15)$;
$H_2 \leftarrow$ **BaseCaseMultiply**$(F_2, G_2, \widetilde{S}, 2p; 1/15)$;
for $i \in [2]$ **do** $H_i \leftarrow H_i \bmod (x^p - 1)$;

$\mathbb{S} \leftarrow$ empty list of integers;
for *every nonzero term* $c_1 x^e \in H_1$ **do**
  $c_2 \leftarrow$ coefficient of degree-e term of H_2;
  if $c_1 \mid c_2$ *and* $\ell \mid (c_2/c_1 - 1)$ *as integers* **then**
    Add $(c_2/c_1 - 1)/\ell \in \mathbb{Z}$ to $\mathbb{S}$;
  else return `fail`;
if $\#\mathbb{S} > \widetilde{S}$ **then return** `fail`;
return $\mathbb{S}$;

4.4 Estimating the sparsity of the product

We now present a step that was not included in [AR15], to address an unlikely but possible case that was missed in the original presentation of the algorithm. If the probabilistic steps of the algorithm succeed, the original algorithm has a guaranteed runtime of $\widetilde{\mathcal{O}}(S\log C + T\log D)$; however, if the algorithm fails to produce the correct sumset $\mathrm{poss}(f, g)$, then we may incorrectly produce a guess for the support $\mathcal{E} \neq \mathrm{supp}(h)$, such that $\#\mathcal{E} \gg T$. In which case our algorithm can exceed the runtime bound of $\widetilde{\mathcal{O}}(S\log C + T\log D)$, conceivably achieving a runtime of $\widetilde{\mathcal{O}}(S(\log C + \log D))$.

Here we probabilistically estimate $T_h = \#h$, in a manner that does not significantly overestimate T_h, similar to how we estimated the cardinality of a sumset in Section 4.3.1. We cannot use the method of **SumsetSize** in order to estimate the size of the produce, because that approach relies on the absence of coefficient cancellation.

We use ideas from Prony's algorithm and linearly generated sequences. Namely, we use the half GCD algorithm in order to compute the minimal generator of the sequence given by h evaluted over a geometric sequence. We evaluate f over powers of a primitive pth root of unity modulo q, for a pair of primes p, q, where $p \mid (q-1)$.

We choose p and q such that probably both p is a good prime for h, and none of the coefficients of h vanish modulo q. This relies on the Monte Carlo subroutine **ProductSparsity**, which probabilistically chooses such a p, q, and a pth root of unity $\omega \in \mathbb{Z}_p$, $\omega \neq 1$.

Procedure ProductSparsity(f, g, S)

Input: Sparse polynomials $f, g \in \mathbb{Z}[x]$; $S > \operatorname{poss}(f, g)$.
Result: With probability at least $2/3$, produces $\widetilde{T_h}$ such that $T_h \leq \widetilde{T_h} \leq 2T_h$. Otherwise produces `fail` or $\widetilde{T_h} < T_h$.
Parameters: $C \geq \|f\|_\infty, \|g\|_\infty$; $D \geq \deg(fg)$; $T_{\text{in}} = \#f + \#g$.

$\lambda \leftarrow \lceil \max(2^{191}, \quad \sqrt{140S\ln(T_{\text{in}}C^2)}, \quad 40(S-1)\ln D \rceil$;
$p, (q, \omega_p) \leftarrow$ **GetPrimeAP-5/6**$(\lambda, (RC^2)^{2R^2}; 1/6)$;
Write f, g as $f = \sum_{i=1}^{\#f} c_{fi} x^{e_{fi}}, g = \sum_{i=1}^{\#g} c_{gi} x^{e_{gi}}$;
$(f_0, g_0) \leftarrow \left(\sum_{i=1}^{\#f} (c_{fi} \bmod q) x^{e_{fi} \bmod p}, \quad \sum_{i=1}^{\#g} (c_{gi} \bmod q) x^{e_{gi} \bmod p} \right)$;
Evaluate $f_0(\omega^i), g_0(\omega^i)$ for all $i \in [0..p)$ via fast multipoint evaluation;
for $i \leftarrow 0$ **to** $2S$ **do** $a_i \leftarrow f_0(\omega^i) g_0(\omega^i)$;
Compute linear generator Φ of $(a_0, \ldots, a_{2S}) \in \mathbb{Z}_q^{2S+1}$ by way of the half GCD algorithm;
return $\widetilde{T_h} = 2\deg(\Phi)$;

Proposition 4.4.1. *With probability at least $2/3$, **ProductSparsity**(f, g, S) produces $\widetilde{T_h} = T_h$. It succeeds with probability at least $2/3$. In the event of failure, or in the case that input S is such that $S < \#\operatorname{poss}(f, g)$, it either produces* `fail` *or $\widetilde{T_h} < T_h$. Its cost is $\widetilde{\mathcal{O}}(T_{\text{in}} \log C + S \log D)$.*

Proof. We first analyze the correctness of **ProductSparsity**. Recall from Lemma 2.4.15 that **GetPrimeAP-5/6**$(\lambda, B; 1/6)$ takes $\lambda, B \in \mathbb{Z}$, where $\lambda \geq 140\sqrt{\ln B}$, and produces a prime $p \in (\lambda..2\lambda]$ chosen uniformly at random, a prime $q \in (2\lambda^2..\lambda^3]$ of the form $q = ap + 1$, and such that q does not divide a fixed unknown integer with absolute value at most B, and a primitive pth root of unity $\omega_p \in \mathbb{Z}_q$, all with probability at least $5/6$.

As the coefficients of fg are trivially each at most $T_{\text{in}}C^2$, and there are at most S of them, the product of the nonzero coefficients of fg are at most $B = (T_{\text{in}}C^2)^S$. As $\lambda \geq 2^{191}$ and

$$\lambda \geq 140\sqrt{S\ln(T_{\text{in}}C^2)} = 140\sqrt{\ln B}$$

our choice of λ satisfies the constraints placed on λ as an input to **GetPrimeAP-5/6**$(\lambda, B; 1/6)$. It follows that with probability at least $5/6$, the subroutine call produces q for which none of coefficients of h vanish modulo q. Assume this is the case, such that $\#(h \bmod q) = \#h$.

As $\lambda \geq 40(S-1)\ln D$, if **GetPrimeAP-5/6** succeeds, it produces a $1/2$-support prime p for h with probability at least $14/15$ by Lemma 3.4.3. If this holds in addition we have that $\#h/2 < \#(h^{\bmod p} \bmod q)$. Assuming $\#h \leq S$, the algorithm produces $\#h^{\bmod p} \bmod q$. Otherwise we produce a minimal generator Φ of degree $\widetilde{T_h} \leq S < T_h$.

Thus, in the case that either the algorithm fails or the input S is incorrect in that $S < \#\mathrm{poss}(f, g)$, either **GetPrimeAP-5/6** produces `fail`, in which case by convention **ProductSparsity** produces `fail`, or **ProductSparsity** produces some value $\widetilde{T_h} < T_h$.

We now analyze the cost. The cost of **GetPrimeAP-5/6**$(\lambda, B; 1/6)$ is

$$\widetilde{\mathcal{O}}(\mathrm{polylog}(\lambda) + \mathrm{polylog}(B)) = \widetilde{\mathcal{O}}(\mathrm{polylog}(ST_{\mathrm{in}} \log C \log D)).$$

The cost of computing $f_0 = f^{\bmod p} \bmod q$ and $g^{\bmod p} \bmod q$ is $\widetilde{\mathcal{O}}(T_{\mathrm{in}}(\log C + \log D + \log q))$ to reduce the coefficients and exponents of f and g. The cost of evaluating f_0 and g_0 at ω^i for all $i \in [0..p)$ via fast multipoint evaluation costs $\widetilde{\mathcal{O}}(p)$ $\mathbb{Z}_q$-operations, or

$$\widetilde{\mathcal{O}}(\sqrt{S} \log C + S \log D) \in \widetilde{\mathcal{O}}(T_{\mathrm{in}} \log C + S \log D)$$

bit operations. Computing the linear generator via the half GCD entails $\widetilde{\mathcal{O}}(S \log q)$ or $\widetilde{\mathcal{O}}(S \cdot \mathrm{polylog}(\log B))$ bit operations. Thus the algorithm costs $\widetilde{\mathcal{O}}(T_{\mathrm{in}} \log C + S \log D)$ bit operations in total. □

4.5 Multiplication with support

We now turn to the problem of multiplying sparse $f, g \in \mathbb{Z}[x]$, provided some set $\mathcal{E}$ such that (probably) $\mathcal{E} \supseteq \mathrm{supp}(fg)$. This algorithm is used twice in our complete polynomial multiplication algorithm: first with a large support set $\mathrm{poss}(f, g)$, but small coefficients, then with the actual support set $\mathrm{supp}(fg)$ but full-size coefficients. We will further suppose that our support set $\mathcal{E}$ contains the support of f and g as well, without affecting the worst-case cost of the algorithm.

4.5.1 Determining the true support of the product $h = fg$

In order to multiply f and g, given some set $\mathcal{E} = \{e_1, \ldots, e_u\} \supseteq \mathrm{supp}(f) \cup \mathrm{supp}(g) \cup \mathrm{supp}(h)$, we will produce the coefficients of $h = \sum_{i=1}^{u} c_i x^i$ as the solution to a transposed Vandermonde

system.

$$\begin{bmatrix} \omega_p^{0\cdot e_1} & \omega_p^{0\cdot e_2} & \cdots & \omega_p^{0\cdot e_u} \\ \omega_p^{1\cdot e_1} & \omega_p^{1\cdot e_2} & \cdots & \omega_p^{0\cdot e_u} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_p^{(u-1)\cdot e_1} & \omega_p^{(u-1)\cdot e_2} & \cdots & \omega_p^{(u-1)} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_u \end{bmatrix} = \begin{bmatrix} f(\omega_p^0)g(\omega_p^0) \\ f(\omega_p^1)g(\omega_p^1) \\ \vdots \\ f(\omega_p^{u-1})g(\omega_p^{u-1}) \end{bmatrix}.$$

over an appropriate ring containing a pth root of unity ω_p. By results on fast multiplication by Vandermonde matrices (e.g. fast multipoint evaluation) and fast Vandermonde system solving (Thm. 2.1.10, and the *Transposition Principle* (Thm. 2.1.11), we can multiply or divide a vector by a $u \times u$ nonsingular transposed Vandermonde matrix over a field K with a cost of $\widetilde{\mathcal{O}}(u)$ K-operations. Lecerf and van der Hoeven also studied fast multiplication and division by Vandermonde matrices in [HL13b], where they also give algorithms for the multiplication of two sparse polynomials, given the support of the product, over a variety of coefficient fields.

In their approach for f, g over $\mathbb{Z}$ (Section 5.4, [HL13b]), the Vandermonde arithmetic is performed modulo $\mathbb{Z}_{p_1 p_2 \ldots p_r}$, where $r \in \mathcal{O}(\log_D C)$ and $\mathbb{Z}_{p_i}$ contains a root of unity ω_i of order exceeding D, for each $i \in [r]$. Our approach differs in that we use pth roots of unity for some good prime $p \ll D$.

Proposition 4.5.1. ***SparseProductCoeffs****$(f, g, \mathcal{E})$ produces a sparse representation of $h = fg$ with probability at least $2/3$. Its cost is $\widetilde{\mathcal{O}}(\#\mathcal{E}(\log C + \log D))$ bit operations. If* ***SparseProductCoeffs*** *fails it produces* `fail`.

Proof. The probabilistic correctness only relies on selection of primes p and the q_i. By Lemma 2.4.16, with probability at least $5/6$, **GetPrimesAP**$(\lambda, 2C_h + 1; 1/6)$ produces a prime $p \in (\lambda..2\lambda]$ and pairs $(q_i, \omega_i), i \in [\ell]$, such that the q_i are distinct primes whose product exceeds $2C_h + 1$, and each $\omega_i \in \mathbb{Z}_{q_i}$ is a pth primitive root of unity. As $\lambda \geq 10S(S-1)\log D$, p is a good prime for $\sum_{e \in \mathcal{E}} x^e$ with probability at least $5/6$. That is, the exponents $e \in \mathcal{E}$ remain distinct modulo p. Thus the algorithm succeeds with probability at least $2/3$.

The cost of reducing every $e \in \mathcal{E}$ modulo q is at most $\widetilde{\mathcal{O}}(\#\mathcal{E} \log D)$. The cost of reducing the coefficients of f and g modulo q is at most $\widetilde{\mathcal{O}}(T_{\text{in}} \log C)$. The cost of multiplying by $\mathbf{V}_{(i)}^{\top}$ and its inverse are both $\widetilde{\mathcal{O}}(\#\mathcal{E})$ operations modulo q_i, or a bit cost of $\widetilde{\mathcal{O}}(\#\mathcal{E} \log q) = \widetilde{\mathcal{O}}(\#\mathcal{E}(\log\log C + \log\log D))$. Thus the algorithm costs $\widetilde{\mathcal{O}}(T_{\text{in}} \log C + \mathcal{E} \log D)$. □

We further remark that the **SparseProductCoeffs** can be made Las Vegas by selecting primes p (line 4) until we discover one which keeps the elements of $\mathcal{E}$ separate modulo p.

Procedure SparseProductCoeffs($f, g, \mathcal{E}$)

Input: Sparse polynomials $f, g \in \mathbb{Z}[x]$; $\mathcal{E} = \{e_1, \ldots, e_S\} \subset \mathbb{Z}_{>0}$, where $\operatorname{supp}(fg), \operatorname{supp}(f), \operatorname{supp}(g) \subseteq \mathcal{E}$.
Output: With probability at least $2/3$, a sparse representation of $h = fg$.
Parameters: $C = \max(\|f\|_1, \|g\|_1)$; $D \geq \max(\deg(fg), \mathcal{E})$; $S = \#\mathcal{E}$.

$C_h \leftarrow S^2C^2$;
Write f, g as $f = \sum_{j=1}^{S} c_j^{(f)} x^{e_j}, g = \sum_{j=1}^{S} c_j^{(g)} x^{e_j}$;
$\lambda \leftarrow \max(21, \quad \lceil 25(S-1)\log D \rceil, \quad \sqrt{42\ln(2C_h+1)})$;
$p, ((q_1, \omega_1), \ldots, (q_\ell, \omega_\ell)) \leftarrow$ **GetPrimesAP**$(\lambda, 2C_h + 1; 1/6)$;
if $e_i \equiv e_j \pmod{p}$ *for some* $i \neq j \in [S]$ **then return** `fail`;
for $j \in [S]$ *and* $\phi \in \{f, g\}$ **do**
  $c_j^{\phi,i)} \leftarrow c_j^{(\phi)} \bmod q_i$, for all $i \in [\ell]$, via multi-modular reduction;
for $i \in [\ell]$ **do**
  for $j \in [S]$ **do** $v_j \leftarrow \omega^{e_j \bmod p} \bmod q_i$;
  $\mathbf{V}_{(i)} \leftarrow \mathbb{V}(v_1, \ldots, v_s)$;
  $\boldsymbol{d}^{(f)} \leftarrow \mathbf{V}_{(i)}^{\top}(c_1^{(f,i)}, \ldots, c_S^{(f,i)}) \in \mathbb{Z}_{q_i}^S$;
  $\boldsymbol{d}^{(g)} \leftarrow \mathbf{V}_{(i)}^{\top}(c_1^{(g,i)}, \ldots, c_S^{(g,i)}) \in \mathbb{Z}_{q_i}^S$;
  $\boldsymbol{d}^{(h)} \leftarrow (d_1^{(f)} d_1^{(g)}, \ldots, d_S^{(f)} d_S^{(g)}) \in \mathbb{Z}_{q_i}^S$;
  $\boldsymbol{c}^{(h,i)} \leftarrow \left(\mathbf{V}_{(i)}^{\top}\right)^{-1} \boldsymbol{d}^{(h)} \in \mathbb{Z}_{q_i}^S$;
for $j \in [S]$ **do**
  Construct $c_j^{(h)} \in [-C_h..C_h]$ by way of Chinese remaindering on the set of congruences $\{\boldsymbol{c}_j^{(h,i)} \bmod q_i : i \in [\ell]\}$;
return $\sum_{j \in S} c_j^{(h)} x^{e_j}$;

4.6 Putting the multiplication algorithm together

We are now in a position to give the entire multiplication algorithm, given by **SparseMultiply**. We then restate and prove Theorem 4.1.2.

Theorem 4.1.2. ***SparseMultiply***(f, g)*, computes a sparse representation of* fg*, with probability at least* $2/3$ *and a bit operation cost of*

$$\widetilde{\mathcal{O}}(S \log D + T \log C).$$

Procedure SparseMultiply(f, g)

Input: Sparse polynomials $f, g \in \mathbb{Z}[x]$.
Output: With probability at least $2/3$, a sparse representation of $h = fg$.
Parameters: $C = \max(\|f\|_\infty, \|g\|_\infty)$; $D = \deg(fg)$; $T_{\text{in}} = \#f + \#g$.

$\mathcal{E} \leftarrow$ **Sumset**$(\text{supp}(f), \text{supp}(g); 1/18)$;
$S \leftarrow \#\mathcal{E}$;

$\widetilde{T_h} \leftarrow$ **ProductSparsity**$(f, g, S; 1/18)$;

$\lambda \leftarrow \max(21, \lceil 30S \log(T_{\text{in}} C^2) \rceil)$;
$q \leftarrow$ **GetPrime**$(\lambda; 1/18)$;
$h_q \leftarrow$ **SparseProductCoeffs**$(f \text{ rem } q, g \text{ rem } q, \mathcal{E}; 1/18)$;
Write h_q as $h_q = \sum_{e \in \mathcal{E}} c_e x^e$;
$\mathcal{E}' \leftarrow \{e \in \mathcal{E} : c_e \bmod q \neq 0\}$;
if $\#\mathcal{E}' > \widetilde{T_h}$ **then return** `fail` ;

return **SparseProductCoeffs**$(f, g, \mathcal{E}'; 1/18)$;

Proof. The probabilistic correctness of **SparseMultiply** relies on the five calls to subroutines, each which fail with probability at most $1/18$, and the selection of the prime q. We require that q does not divide any of the coefficients of h. As $\#h \leq S$ and $\|h\|_\infty \leq T_{\text{in}} C^2$, the product of the nonzero coefficients of h, call it Π, has absolute value at most $(T_{\text{in}} C^2)^S$. By Lemma 2.4.4, $(\lambda..2\lambda]$ contains at least $3\lambda/5 \ln \lambda \geq \log_\lambda \left(T_{\text{in}} C^2\right)/18)$ primes, such that q chosen uniformly at random from $(\lambda..2\lambda]$ divides Π is at most $1/18$. Thus the algorithm succeeds with probability at least $2/3$ as desired.

The cost is dominated by the calls to **SparseProductCoeffs**, which by Proposition 4.5.1 cost

$$\widetilde{\mathcal{O}}(S(\log \lambda + \log D) + T(\log C + \log D)) = \widetilde{\mathcal{O}}(S \log C + T \log D).$$

□

4.6.1 Extensions to Laurent polynomials and various coefficient rings

Our multiplication algorithm easily extends to Laurent polynomials. Given $f, g \in \mathsf{R}[x^{\pm 1}]$ with exponents $e \in [-D..D]$, we can instead naively compute the polynomial product $h' = (x^D f)(x^D g)$ with cost $(S \log D + T \log C)$ by Theorem 4.1.2. The products $x^D f, x^D g$, and $h = x^{-2D} h$ can be computed by adding or subtracting from the exponents of f, g, and h with cost $\widetilde{\mathcal{O}}()$. We can further reduce the multiplication of n-variate Laurent polynomials with $\text{absdeg}(h) = D$ to a

univariate product with absolute degree $\mathcal{O}(D^n)$ via Kronecker substitution. This reduction is also bounded by the cost of multiplication. We state this as a Corollary.

Corollary 4.6.1. *Given $f, g \in \mathbb{Z}[x_1^{\pm 1}, \ldots, x_n^{\pm 1}]$, with $\operatorname{supp}(fg) \subset [-D..D]^n$, one can compute fg with probability at least $2/3$ and cost $\widetilde{\mathcal{O}}(Sn \log C + T \log D)$ bit operations.*

In other coefficient rings, we can still efficiently compute $\operatorname{poss}(f, g)$; however, we cannot as easily decouple the factor $\log C$ from S. To compute $f, g \in \mathbb{Z}_m$, we can embed the problem in $\mathbb{Z}$ with bound $C = m/2$, and reduce the resulting product modulo m.

Multiplying $f, g \in \mathbb{F}_{p^v}[x]$, where p is prime, is more involved. We suppose $\mathbb{F}_{p^v}$ is represented as $\mathbb{Z}_p[y]/\langle \Phi \rangle$ for some irreducible polynomial $\Phi \in \mathbb{Z}_p[y]$ of degree v. We let $f', g' \in \mathbb{Z}[y, x]$ be the embedding of $f, g \in \mathbb{Z}[y, x]$. Then $h' = f'g'$ has $\deg_y(fg) \leq 2s - 2$ and is $\mathcal{O}(vS)$-sparse. Thus we can obtain h' from the product $h'' = f'(y, y^{2v-2})g'(y, y^{2v-2})$ with degree $\mathcal{O}(vD)$ and at most $\mathcal{O}(vS)$ terms, and the heights of f' and g' bound by p. We can extract $h = fg$ by reducing h' modulo p and $\Phi(y)$. As h' has height at most $T_{\text{in}}p^2$, this costs $vS \log p$. Generalizing to multivariate Laurent polynomials gives us the following.

Corollary 4.6.2. *There exists a Monte Carlo algorithm that, given $f, g \in \mathbb{F}_q[x_1^{\pm 1}, \ldots, x_m^{\pm 1}]$, with $\operatorname{supp}(fg) \subset [-D..D]^n$ and q a prime power, computes a sparse representation of fg with probability at least $2/3$ and cost $\widetilde{\mathcal{O}}(S(\log q + n \log D))$.*

4.7 Conclusions and future work

We presented new probabilistic and asymptotically fast algorithms to compute the sumset of two finite sets of integers, and to multiply two polynomials with integer coefficients. There are many open problems that remain. The biggest open problem is to remove all quadratic factors from a multiplication algorithm. A Prony-based method, even ignoring precomputation, appears to require $\widetilde{\Omega}(\log^2 D)$ bit operations; whereas both the Cole–Hariharan algorithm and the new method given by **SparseMultiply** admit a soft-oh cost factor of S, which in the worst case is $\widetilde{\Omega}(T^2)$.

Problem 4.7.1. *Give an algorithm which computes fg for arbitrary $f, g \in \mathbb{Z}[x]$ with worst-case runtime subquadratic in the bit sizes of f, g, and h.*

We do not even know whether we can certify a polynomial product in softly-linear time. Namely, we pose the following problem.

Problem 4.7.2. *Given $f, g, \phi \in \mathbb{Z}[x]$ and $C > \|f\|_\infty, \|g\|_\infty, \|\phi\|_\infty$, test whether $fg = \phi$ with cost $\widetilde{\mathcal{O}}((\#f + \#g + \#\phi)(\log C + \log D))$, either probabilistically or deterministically.*

A Las Vegas or deterministic solution to that problem would allow us to make **SparseMultiply** a Las Vegas algorithm.

How could they see anything but the shadows if they were never allowed to move their heads?

Plato, *The Allegory of the Cave*

Chapter 5

Multivariate sparse interpolation

5.1 Introduction

We now bring our attention to the probabilistic interpolation of sparse *multivariate* polynomials. In this setting we suppose f is of the form

$$f = \sum_{i=1}^{t} c_i x_1^{e_{i1}} x_2^{e_{i2}} \dots x_n^{e_{in}} \in \mathsf{R}[x_1, \dots, x_n], \tag{5.1}$$

where the exponents $\boldsymbol{e}_i = (e_{i1}, \dots, e_{in})$ are sorted lexicographically. Throughout this chapter we will use the parameters n for the number of variables, $D \geq \max_{i=1}^{n} \deg_{x_i} f$ for the partial degrees of f, and $T \geq \#f$ for the sparsity.

We first describe Zippel's probabilistic multivariate interpolation algorithm for a black-box polynomial over a finite field, which reduces the problem to univariate interpolation. We then present new algorithms for multivariate sparse interpolation, based on a technique we call *randomized Kronecker substitutions*, which we liken to Kronecker substitution. The key idea therein is to construct a multitude of univariate images of reasonably small degree, whereby most terms of f do not collide in each image. We give two methods of substitution: one for general n-variate polynomials, and one strictly for bivariate polynomials that is advantageous in the case that $\sqrt{T} \ll \ln D$.

These methods of substitution give rise to a variety of algorithms. In Section 5.4, we present an algorithm for a black-box polynomial f over an arbitrary ring, that follows a similar template to iterative interpolation (Sec. 3.4). In Section 5.6, we give a multivariate analogue of majority-vote interpolation (Sec. 3.5) for a black-box polynomial over a sufficiently large finite field. These algorithms rely on a univariate interpolation procedure as a subroutine, and their costs are

expressed partly in terms of these subroutine calls. This algorithm appeared in [AR14] and was joint work with Daniel S. Roche.

A caveat of this multivariate majority-rule interpolation is that it requires a linear system to be solved in order to construct every term of f. In Section 5.7, we give a procedure for the interpolation of a highly multivariate (which we will fondly refer to as **ultravariate**) extended black-box polynomial, that outperforms a combination of majority-rule multivariate and univariate interpolation, in the case that the cost of the latter approach is dominated by solving $n \times n$ linear systems. It achieves this improvement by reducing the linear algebra overhead via structured linear systems. This algorithm appears in [AGR15], respectively, and is joint work with Daniel S. Roche and Mark Giesbrecht.

We introduce some multivariate notation. We will often write (5.1) as

$$f(\boldsymbol{x}) = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{R}[\boldsymbol{x}]. \tag{5.2}$$

We will also consider univariate images of f of the form

$$f(z^{\boldsymbol{s}}) \stackrel{\text{def}}{=} f(z^{s_1}, \ldots, z^{s_n}) \in \mathsf{R}[z],$$

where $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{N}^n$. For a vector $\boldsymbol{a} \in \mathsf{R}^n$, we will also use throughout the notation

$$\begin{aligned} f(\boldsymbol{a}z^{\boldsymbol{s}}) &\stackrel{\text{def}}{=} f(a_1 z^{s_1}, \ldots, a_n z^{s_n}), \\ f(\boldsymbol{a}\boldsymbol{x}) &\stackrel{\text{def}}{=} f(a_1 x_1, \ldots, a_n x_n), \\ f(\boldsymbol{a}\boldsymbol{x}^{\boldsymbol{s}}) &\stackrel{\text{def}}{=} f(a_1 x_1^{s_1}, \ldots, a_n x_n^{s_n}). \end{aligned}$$

This chapter will revisit *sparsity-preserving maps*, and so we generalize them in the natural way to a multivariate setting.

Definition 5.1.1. *We say $\Phi : \mathsf{R}[x_1, \ldots, x_n] \to \mathsf{R}[y_1, \ldots, y_m]$ is a* **sparsity-preserving map** *if Φ is R-linear and there exists a map $\Phi^* : \mathbb{N}^n \to \mathbb{N}^m$ such that Φ is given by $\boldsymbol{x}^{\boldsymbol{e}} = \boldsymbol{y}^{\Phi^*(\boldsymbol{e})}$. We call Φ^* the* **induced-map** *of f. We say the preimage σ of a formal term $c\boldsymbol{y}^{\boldsymbol{d}}$ of $\Phi(f)$ to the sum of all terms of f that map to a term with exponent $\boldsymbol{d}$ under the map Φ, i.e., for $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i}$,*

$$\sigma = \sum_{\substack{i \in [t] \\ \Phi^*(\boldsymbol{e}_i) = \boldsymbol{d}}} c_i \boldsymbol{x}^{\boldsymbol{e}_i}.$$

We extend definitions of **singletons**, **colliding terms**, **colliding exponents** and **collisions** in the natural way.

The algorithms herein rely on constructing sets of univariate images.

Definition 5.1.2. *Let $f \in \mathsf{R}[\boldsymbol{x}]$ be a black-box polynomial. We define a d-***query** *or* **query of degree** *d for f to be a univariate image of the form $f(z^{\boldsymbol{s}})^{\bmod d} \in \mathsf{R}[z]$ of degree at most $d-1$, where $\boldsymbol{s} \in \mathbb{N}^n$.*

We let a **shifted** *d-***query** *of f denote an image of the form $f(\boldsymbol{a}z^{\boldsymbol{s}})^{\bmod d}$ we call $\boldsymbol{a}$ the* **shift** *of the query. If $\mathbf{a}$ is selected from S^n for a ring extension $\mathsf{S} \supseteq \mathsf{R}$, we may call the shifted d-query a* S-shifted d-query.

A d-query is a generalization of a p-query. Namely, if f is a univariate extended black-box polynomial, then a p-query for f is a d-query with $p = d$. We relax the condition that f is an extended black-box polynomial, because in some cases we may not need access to roots of unity taking a large variety of prime orders. Sometimes a d-query will be unreduced, i.e., we say $f(\boldsymbol{a}z^{\boldsymbol{s}})$ is a d-query for $d = \deg(f(\boldsymbol{a}z^{\boldsymbol{s}})) + 1$. We further remark that a query of a query of f is itself a query of f. We may informally refer to queries also as **substitutions** or **images**. Generally we will suppose the cost of such a query is at least the bit size of the sparse representation of the query. In the case that f is given by an SLP, it suffices to perform arithmetic in $\mathsf{R}[z]^{\bmod d}$, which gives the following cost:

Lemma 5.1.3. *Given a length-L SLP computing $f \in \mathsf{R}[x]$, we can compute a d-query of f with a cost of $\widetilde{\mathcal{O}}(Ld)$ ring operations.*

We remark in this chapter that we do not consider the selection of primes in our probabilistic analysis. We merely remark that we can construct n primes from $(\lambda..2\lambda]$ with probability exceeding $1-\mu$ in $\widetilde{\mathcal{O}}(n\log(\mu^{-1}) \cdot \operatorname{polylog}(\lambda))$ bit operations, e.g., using GetPrimes. This simplifies the analysis of a number of Monte Carlo procedures herein, without affecting the correctness. Moreover, many of the algorithms in this chapter rely on univariate interpolation algorithms. In our analysis we suppose our univariate interpolation subroutines are deterministic; however, it would be straightforward to extend these multivariate procedures to allow for Monte Carlo univariate interpolation algorithms.

5.1.1 Comparison of sparse multivariate algorithms

Table 5.1 compares various interpolation and testing algorithms that work via reduction to univariate interpolation or identity testing. We mention a randomized multivariate polynomial identity test due to Klivans and Spielman [KS01]. They attempted to minimize the number of random bits required by their identity test, while keeping it polynomial time. They use $\mathcal{O}(\log(nTD))$ bits of randomness.

Kronecker substitution gives a deterministic interpolation or testing procedure, provided it uses a deterministic univariate interpolation/testing subroutine. We describe Zippel's algorithm in detail in the subsequent section.

Algorithm	Section	Ring	# of queries	degree
Kronecker	2.6	R	1	$(D+1)^n$
Zippel [Zip79]	5.2	$\mathbb{F}_q$	nt	D
multivariate iterative	5.4	R	$\mathcal{O}(\log^2 T)$	$\mathcal{O}(nDT)$
			$\mathcal{O}(n \log T)$	$\widetilde{\mathcal{O}}(DT(nD+T))$
bivariate majority rule	5.6	$\mathbb{F}_q$	$\mathcal{O}(\log^2 T)$	$\mathcal{O}(D \log D\sqrt{T})$
multivariate majority rule	5.6	$\mathbb{F}_q$	$\mathcal{O}(n + \log T)$	$\mathcal{O}(nTD)$
ultravariate majority rule	5.7	$\mathbb{F}_q$	$\widetilde{\mathcal{O}}(n \log T(\log D + \log T))$	$\widetilde{\mathcal{O}}((T+n)\log D)$
Klivans–Spielman PIT [KS01]		$\mathbb{F}_q$	n	$\mathcal{O}(n^2T^2D)$

Table 5.1: Algorithms for sparse multivariate interpolation and testing via univariate images

5.2 Zippel's algorithm

Any discussion of multivariate polynomial interpolation would be incomplete without Zippel's algorithm. In [Zip79], Richard Zippel gave a probabilistic algorithm for the interpolation of a sparse multivariate black-box polynomial f over a field $\mathsf{K} = \mathbb{F}_q$ or $\mathbb{Q}$. We analyze these cases, though the algorithm can extend to any ring containing a sufficiently large *regular-difference set*. As K can be infinite, we let $\mathcal{R}$ denote a regular-difference set in K (e.g. any finite subset of K). The algorithm does not require an a priori bound T on the sparsity of the interpolant f, though it can use such a bound to lower its cost. We describe a probabilistic version of the algorithm as given in [Zip90].

To explain Zippel's algorithm we will define polynomials whose coefficients are themselves polynomials. In Section 5.2 we reserve B and C to denote "polynomial coefficients" and b and c to denote field constants.

Zippel's algorithm constructs $f \in \mathsf{K}[x_1, \ldots, x_n]$ from a hierarchy of multivariate images. The algorithm learns the support of f with respect to increasingly many variables. For $k \in [n]$, write f as

$$\begin{gathered} f^{(k)} = \sum_{i=1}^{t^{(k)}} C_i^{(k)} \boldsymbol{x}^{\boldsymbol{e}_i^{(k)}} \in \mathsf{K}[x_{k+1}, \ldots, x_n][x_1, \ldots, x_k], \quad \text{where} \\ 0 \neq C_i^{(k)} \in \mathsf{K}[x_{k+1}, \ldots, x_n], \quad \boldsymbol{e}_i^{(k)} \in \mathbb{Z}_{\geq 0}^k, \quad \text{and} \quad \boldsymbol{x}^{\boldsymbol{e}_i^{(k)}} = \prod_{j=1}^{k} x_j^{e_{ij}^{(k)}} \quad \text{for} \quad i \in [t^{(k)}]. \end{gathered} \tag{5.3}$$

We define $\operatorname{supp}_k(f) = \{\boldsymbol{e}_i^{(k)} : i \in [t_k]\}$. Observe that for $m > k$, $\operatorname{supp}_m(f)$ determines $\operatorname{supp}_k(f)$. For $k < m \in [n]$, $\operatorname{supp}_k(f) = \{(e_1^{(m)}, \ldots, e_k^{(m)}) : e^{(m)} \in \operatorname{supp}_m(f)\}$. In other words, $\operatorname{supp}_k(f)$ is comprised of the set of length-k truncations of all length-m exponents in $\operatorname{supp}_m(f)$. It follows

that $t^{(1)} \leq t^{(2)} \leq \cdots \leq t^{(n)} = \#f$. Zippel's algorithm learns $\operatorname{supp}_k(f)$ for increasing k, until we have $\operatorname{supp}_n(f) = \operatorname{supp}(f)$.

Zippel's algorithm initially chooses a set of random evaluation points $\omega_{i0} \in \mathcal{R}$, $i \in [n]$. We also define the images

$$f_0^{(\ell)} \stackrel{\text{def}}{=} f(x_1, \ldots, x_\ell, \omega_{\ell+1,0}, \ldots, \omega_{n0}) \in \mathsf{K}[x_1, \ldots, x_\ell], \quad \text{for } \ell \in [n]. \tag{5.4}$$

In the first stage of the algorithm, Zippel's algorithm densely interpolates the univariate image $f_0^{(1)}$. The algorithm makes a (not necessarily random) choice of $D+1$ distinct values $\omega_{11}, \ldots, \omega_{1D} \in \mathsf{K} \setminus \{\omega_{10}\}$, and evaluates $f_0^{(1)}(\omega_{1j})$ for $j \in [0..D]$, from which it interpolates $f_0^{(1)}$.

We describe the kth stage of the algorithm, where $k \in (1..n]$. In the kth stage we always evaluate f at a point $\boldsymbol{x}$ where $x_\ell = \omega_\ell$ for all $\ell > k$. We write

$$\begin{gathered} f_0^{(k-1)}(x_1, \ldots, x_{k-1}) = \sum_{i=1}^{t'} c_{i0} \boldsymbol{x}^{\boldsymbol{e}_i}, \quad \text{where} \\ t' = t^{(k-1)}, \quad c_{i0} = C_i^{(k-1)}(\omega_{k0}, \omega_{k+1,0}, \ldots, \omega_{n0}), \quad \text{and} \quad \boldsymbol{e}_i = \boldsymbol{e}_i^{(k-1)} \quad \text{for} \quad i \in [t']. \end{gathered} \tag{5.5}$$

At the start of the k-th stage, we have $f_0^{(k-1)}$, and we aim to learn $f_0^{(k)}$. Zippel's algorithm supposes that each of these c_{i0} are nonzero. In other words, Zippel's algorithm assumes that $\operatorname{supp}\left(f_0^{(k-1)}\right) = \operatorname{supp}_{k-1}(f)$, such that we can ascertain the latter from the former. Zippel gives probabilistic guarantees on this assumption, under which, and given the expression (5.5) for $f_0^{(k-1)}$, we can write $f_0^{(k)}$ as

$$f_0^{(k)}(x_1, \ldots, x_k) = \sum_{i=1}^{t'} C_i' \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{K}[x_k][x_1, \ldots, x_{k-1}], \tag{5.6}$$

where $C_i' \in \mathsf{K}[x_k]$ and $C_i'(\omega_{k0}) = c_{i0}$ for $i \in [t']$. We will densely interpolate the "coefficients" C_i' which determine $f_0^{(k)}$. We choose distinct values $\omega_{k1}, \omega_{kn} \ldots, \omega_{kD} \in \mathsf{K} \setminus \{\omega_{k0}\}$, and compute the images

$$f_j^{(k-1)} \stackrel{\text{def}}{=} f_0^{(k)}(x_1, \ldots, x_{k-1}, \omega_{kj}) = \sum_{i=0}^{t'} c_{ij} \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{K}[x_k], \quad j \in [0..D],$$

We use sparse interpolation in order to construct the j-th image $f_j^{(k-1)}$. Under the algorithm's aforementioned assumption, $\operatorname{supp}\left(f_j^{(k-1)}\right) \subseteq \operatorname{supp}\left(f_0^{(k-1)}\right)$, in which case we need only determine the coefficients $c_{ij}^{(k-1)}$, for $i \in [t']$. We choose $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_{k-1}) \in \mathcal{R}^{k-1}$ such that the values $\boldsymbol{\zeta}^{\boldsymbol{e}_i}$ are distinct for $i \in [t']$. We make the evaluations

$$a_{ij} = f^{(k)}(\zeta_1^i, \ldots, \zeta_{k-1}^i, \omega_{kj}), \quad i \in [t'], j \in [0..D]. \tag{5.7}$$

Algorithm 15: Zippel's algorithm

Input: A black-box polynomial $f \in \mathsf{K}[x_1, \ldots, x_n]$; $D \geq \max_{i=1}^{n} \deg_{x_i}(f)$.
Output: With high probability, f (See Section 5.2.1 for analysis).

Fix a finite regular-difference set $\mathcal{R} \subset \mathsf{K}$;
Randomly select $\omega_{10}, \ldots, \omega_{n0} \in \mathcal{R}$;

Choose D distinct values $\omega_{11}, \ldots, \omega_{1D} \in \mathsf{K} \setminus \{\omega_{10}\}$;
Compute $f_0^{(1)}$ from evaluations $f_0^{(1)}(\omega_{1j})$, $j \in [0..D]$;

for $k \leftarrow 2$ **to** n **do**
  Write $f_0^{(k-1)} = \sum_{i=1}^{t'} c_i x^{\boldsymbol{e}_i} \in \mathsf{K}[x_1, \ldots, x_{k-1}]$ in terms of its sparse representation.
  Choose D distinct values $\omega_{k1}, \ldots, \omega_{kD}$ from $\mathsf{K} \setminus \{\omega_{k0}\}$;
  Choose $\zeta_1, \ldots, \zeta_t \in \mathcal{R}$, such that the values $\boldsymbol{\zeta}^{\boldsymbol{e}_i}$, $i \in [t']$, are distinct;
  for $j \leftarrow 0$ **to** D **do**
    for $i \leftarrow 1$ **to** t' **do** $a_{ij} \leftarrow f(\zeta_1^i, \ldots, \zeta_{k-1}^i, \omega_{kj}, \omega_{k+1,0}, \ldots, \omega_{n0})$;
    $(c_{1j}, \ldots, c_{t'j}) \leftarrow$ solution $\boldsymbol{c}$ to system $\left[\zeta^{(i-1)e_{ij}}\right]_{i,j=1}^{t'} \boldsymbol{c} = (a_{i1}, \ldots, a_{it'})$;
  for $i \leftarrow 1$ **to** t' **do**
    Densely interpolate $C_i' \in \mathsf{K}[x_k]$ from evaluations $C_i'(\omega_{kj}) = c_{ij}, j \in [0..D]$;
  $f_0^{(k)} \leftarrow \sum_{i=1}^{t'} C_i' \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{K}[x_k][x_1, \ldots, x_{k-1}]$;
return $f_0^{(n)} = \sum_{i=1}^{t} c_{i0} x^{\boldsymbol{e}_i} \in \mathsf{K}[x_1, \ldots, x_n]$;

For each $j \in [0..D]$, this gives a $t' \times t'$ Vandermonde system

$$\begin{bmatrix} \boldsymbol{\zeta}^{0\boldsymbol{e}_1} & \boldsymbol{\zeta}^{0\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^{0\boldsymbol{e}_{t'}} \\ \boldsymbol{\zeta}^{1\boldsymbol{e}_1} & \boldsymbol{\zeta}^{1\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^{1\boldsymbol{e}_{t'}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\zeta}^{(t'-1)\boldsymbol{e}_1} & \boldsymbol{\zeta}^{(t'-1)\boldsymbol{e}_2} & \cdots & \boldsymbol{\zeta}^{(t'-1)\boldsymbol{e}_{t'}} \end{bmatrix} \begin{bmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{t'j} \end{bmatrix} = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{t'j}, \end{bmatrix} \tag{5.8}$$

whose solution determines the desired values c_{ij}, $i \in [t']$. After the nth stage we have $f_0^{(n)} = f \in \mathsf{K}[x_1, \ldots, x_n]$. Algorithm 15 describes the approach.

5.2.1 Analysis of Zippel's algorithm

We first analyze the correctness of Zippel's algorithm. Here we let $t = \#f$. In the kth stage, $k > 1$, we require that $C_i^{(k-1)}(\omega_{k0}, \ldots, \omega_{n0}) \neq 0$ for all $i \in [t^{(k-1)}]$. If the ω_{k0} are chosen uniformly at

random from a finite set $\mathcal{R}$, by the DLSZ Lemma, the probability that this occurs for a fixed $C_i^{(k-1)}$ is at least $1 - \deg(C_i^{(k-1)})/\#\mathcal{R}$. As each $c_i^{(k-1)}$ has total degree at least $(n-k+1)D$, by the union bound, the probability that this occurs for $C_i^{(k-1)}$ for every $k \in (1..n]$ and $i \in [t^{(k-1)}]$ is at least $1 - \mu$, where

$$\mu = \frac{1}{\#\mathcal{R}} \sum_{k=2}^{n} \sum_{i=1}^{t^{(k-1)}} \deg(C_i^{(k-1)}) \leq \frac{1}{\#\mathcal{R}} \sum_{k=1}^{n-1} ktD \leq \frac{1}{2\#\mathcal{R}} n^2 tD.$$

We require, in addition, that in the kth stage, the values $\boldsymbol{\zeta}^{\boldsymbol{e}^{(k-1)_i}}$ are distinct for $1 \leq i \leq t^{(k-1)}$, such that the resulting Vandermonde systems are nonsingular. In the case when $\mathsf{K} = \mathbb{Q}$, we can choose ζ_j to be the jth prime, such as in the Ben-Or–Tiwari algorithm. This gives the following in the case that $\mathsf{K} = \mathbb{Q}$.

Proposition 5.2.1. *Suppose, for t-sparse $f \in \mathbb{Q}[x_1, \ldots, x_n]$ and $D \geq \max_i \deg_{x_i} f$, that Algorithm 15 chooses $\omega_{10}, \ldots, \omega_{n0}$ independently and uniformly at random from a finite set $\mathcal{R} \subset \mathbb{Q}$, and takes ζ_i to be the ith prime number. Then the algorithm takes $\mathcal{O}(ntD)$ black-box queries to f and interpolates f with probability at least $1 - n^2tD/(2\#\mathcal{R})$.*

In the case that $\mathsf{K} = \mathbb{F}_q$, where $q > (D+1)^t$, we could choose ζ of multiplicative order $(D+1)^t$ and let $\zeta_j = \zeta^{(D+1)^{j-1}}$ for all $j \in [t^{(k-1)}]$. This essentially maps the $(k-1)$-variate exponents $\boldsymbol{e}_i^{(k-1)}$ to univariate exponents of the form $e_i = \sum_{j=1}^{k-1} x_j^{e_{ij}^{(k-1)}}$ via Kronekcer substitution such that, given the order of ζ, the values $\boldsymbol{\zeta}^{\boldsymbol{e}_i}$ are distinct. This, however, requires access to a very high-order element. We can instead use Proposition 2.7.8, which guarantees that, for a finite set $\mathcal{R} \subset \mathbb{F}_q$ and $\boldsymbol{\zeta} \in \mathcal{R}^{t'}$ chosen uniformly at random, that the Vandermonde system 5.8 is singular with probability at most $\frac{1}{2}t'(t'-1)D/\#\mathcal{R} < \frac{1}{2}t^2D/\#\mathcal{R}$. Accounting for all $n-1$ Vandermonde systems gives the following.

Proposition 5.2.2. *Suppose, for t-sparse $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, Algorithm 15 chooses $\omega_{10}, \ldots, \omega_{n0}$ and $\zeta_1, \ldots, \zeta_n$ independently and uniformly at random from $\mathbb{F}_q$. Then the algorithm takes $\mathcal{O}(ntD)$ black-box queries to f and succeeds with probability exceeding*

$$1 - \frac{nt(n+t)D}{2q}.$$

We note that if $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ is given by an extended black-box, then we can choose $u = \lceil \log_q(nD^2t^2\mu^{-1}) \rceil$ and choose random evaluation points from $\mathbb{F}_{q^u}$, such that the algorithm succeeds with probability at least $1 - \mu$.

In the case that if f is a black-box polynomial over $\mathbb{F}_q$, then it suffices that

$$q \geq (1+\epsilon) \max(\tfrac{1}{2}t^2D, nD),$$

for a constant $\epsilon > 0$. Then the n Vandermonde systems are individually nonsingular with probability at least $\frac{\epsilon}{1+\epsilon}$. A good heuristic would be in every stage to reselect $\boldsymbol{\zeta}$ until we get a nonsingular system. Alternatively, in order to keep the algorithm Monte Carlo, we can select at most $\lceil \ln(6n)(1+\epsilon)/\epsilon \rceil$ vectors $\boldsymbol{\zeta} \in \mathbb{F}_q^n$, choosing one that gives a nonsingular Vandermonde system, such that with probability exceeding $1/(1+\epsilon)^{\ln(6n)(1+\epsilon)/\epsilon} \leq \exp(-\ln(6n)) = 1/(6n)$, we have a nonsingular Vandermonde system at the kth stage of the algorithm. We thus have nonsingularity at every stage with probability at least $1/6$.

We can similarly choose multiple tuples of the form $\boldsymbol{\omega}_0 = (\omega_{10}, \ldots, \omega_{n0})$ and then assume the support $\mathrm{supp}_{k-1}(f)$ is the union of $\mathrm{supp}(f_0^{(k-1)})$ given by (5.4), over all choices of $\boldsymbol{\omega}$. Each of the $C_i^{(k-1)}$ has total degree at most nD, such that $C_i^{(k-1)}$ evaluates to zero with probability at most $\frac{1}{1+\epsilon}$. We can similarly choose $\lceil \ln(6nt)(\epsilon+1)/\epsilon \rceil$ vectors $\boldsymbol{\omega}$, such that each of the at-most nt coefficient polynomials $C_i^{(k-1)}$, for $k \in [2..n]$ and $i \in [t^{(k-1)}]$ are nonzero for some choice $\boldsymbol{\omega}$, with probability at least $5/6$.

If the algorithm does not start with knowledge of t, and there is no supplied bound $T \geq t$, one can naively take $T = D^n$, given the degree bound D.

5.2.2 An alternative formulation of Zippel's algorithm

We note that we could alternatively reverse the order of dense and sparse interpolation in Zippel's algorithm. We mention this to more directly compare Zippel's algorithms to other interpolation methods whose cost we measure in terms of the univariate images required. Specifically, in the kth stage of Zippel's algorithm, one could use dense interpolation on the evaluations $a_{i0}, \ldots, a_{iD}$ given by (5.7) in order to construct the univariate images

$$g_i^{(k-1)} \stackrel{\text{def}}{=} f(\zeta_1^i, \ldots, \zeta_{k-1}^i, x_k, \omega_{k+1,0}, \ldots, \omega_{n0}) = \sum_{j=0}^{D} b_{ij} x_k^j \in \mathsf{K}[x_k], \quad i \in [t'],$$

where $t' = t^{(k-1)}$. We can write $f_0^{(k)}$ defined by (5.4) as

$$f_0^{(k)} = \sum_{j=0}^{D} B_j^{(k)} x_k^j \in \mathsf{K}[x_1, \ldots, x_{k-1}][x_k],$$

where $B_j^{(k)} \in \mathsf{K}[x_1, \ldots, x_{k-1}]$. Note the union of $\mathrm{supp}(B_j^{(k)})$, for $j \in [0..D]$, is $\mathrm{supp}_{k-1}(f)$. Thus to interpolate $g_i^{(k)}$, one only needs find the coefficients of each of the $B_j^{(k)}$. This again can be

accomplished via the Vandermonde system

$$\begin{bmatrix} \zeta^{0e_1} & \zeta^{0e_2} & \dots & \zeta^{0e_t} \\ \zeta^{1e_1} & \zeta^{1e_2} & \dots & \zeta^{1e_t} \\ \vdots & \vdots & \ddots & \vdots \\ \zeta^{(t'-1)e_1} & \zeta^{(t-1)e_2} & \dots & \zeta^{(t'-1)e_t} \end{bmatrix} \boldsymbol{b}_j = \begin{bmatrix} b_{0j} \\ b_{1j} \\ \vdots \\ b_{t-1,j} \end{bmatrix},$$

whose solution $\boldsymbol{b}_j \in \mathsf{K}^{t'}$ gives representation $B_j^{(k)} = \sum_{i=1}^{t'} b_{ji} \boldsymbol{x}^{\boldsymbol{e}_i}$. Clearly we need only do this for j such that $(b_{0j}, \dots, b_{t-1,j})$ is nonzero. In this way, we see that Zippel's algorithm constructs at most nt D-queries: the $f_i^{(k-1)}$ for $k \in (1..n]$ and $i \in [t^{(k)}]$, as well as the images produced in the first stage.

5.3 Randomized Kronecker substitutions

In this section we describe a multivariate interpolation strategy first given in [AR14]. This is joint work with Daniel S. Roche. As in the case of Kronecker substitution, this technique relies on univariate interpolation as a subroutine. We reconstruct a polynomial from a set of univariate images, i.e., substitutions. As such, the method works in any setting over any coefficient ring whereby one can interpolate T-sparse univariate polynomial of appropriately large degree. Such settings include straight-line programs and extended black-boxes over arbitrary commutative rings with identity, and black-box polynomials over integral domains of appropriately large cardinality.

Recall that Kronecker substitution creates a single $(D+1)^n$-query of f. We will instead choose tuples of n integers $s_1, \dots, s_n$ at random and construct multiple queries $g(z) = f(z^{s_1}, \dots, z^{s_n})$. When the s_i are not too large (e.g., if $s_i \ll D^{n-1}$), then the degree of the univariate image will be appreciably less than that resulting from Kronecker substitution.

Our method poses technical obstacles. First, two or more distinct terms in f may collide in g. Second, as with sparse univariate interpolation, we need a means of reconstructing the multivariate exponents from multiple queries. We show how choosing $\mathcal{O}(n + \log T)$ such random queries may overcome these difficulties.

Example 5.3.1. *Consider, for example, the bivariate polynomial*

$$f = 3x^9y - 2x^5y^4 - y^6 + x^2y^9.$$

Using degree bound $D = 10$, the resulting Kronecker substitution would create a substitution

$$g(z) = f(z, z^{10}) = 3z^{19} - 2z^{45} - z^{60} + z^{92}.$$

Every term from g comes from a single term in f. Suppose instead we make substitutions

$$f_1 = f(z^5, z^2) = -z^{12} + z^{28} - 2z^{29} + 3z^{47}$$
$$f_2 = f(z^2, z^5) = 3z^{23} - 3z^{30} + z^{49}.$$

For the first substitution we have that $\#f_1 = \#f$, such that there were no term collisions. In the second substitution f_2, the two terms $-2x^5y^4$ and $-y^6$ collided to produce the term $-3z^{30}$.

For the two terms not involved in a collision, both images can be used to recover the original terms in f. The two terms in f_1 and f_2 with coefficient 1 have degrees 47 and 23, producing a linear system

$$\begin{bmatrix} 5 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} 28 \\ 49 \end{bmatrix},$$

whose solution $(e_1, e_2) = (2, 9)$ recovers the original term x^2y^9.

In the next sections we determine an appropriate choice of substitution entries s_i, such that a suitably large number of terms of f do not collide in each substitution. We give two substitution techniques: one for the strictly bivariate case that is advantageous in the case that $T \gg \log D$, and one for the general n-variate case, where $n \geq 2$.

5.3.1 Bivariate substitutions

Throughout this section, we will assume $f \in \mathsf{R}[x, y]$ is an unknown bivariate polynomial written as

$$f = \sum_{i=1}^{t} c_i x^{u_i} y^{v_i}.$$

We further assume partial degree bounds $D_x \geq \deg_x(f)$, $D_y \geq \deg_y(f)$, in addition to $T \geq t$. Our approach is to choose substitutions of the form

$$g(z) = f(z^p, z^q),$$

for randomly chosen primes p and q. In order to probabilistically bound the number of collisions that occur for a choice of substitution vector, we prove the following lemma:

Lemma 5.3.2. *Let $f \in \mathsf{R}[x, y]$ with partial degrees at most D_x, D_y and at most $T \geq 2$ nonzero terms, and $\mu \in (0, 1)$. Fix a nonzero term τ of f and define*

$$M = \tfrac{25}{9}(T-1)\mu^{-1} \ln D_x \ln D_y,$$
$$\lambda_x = \max\left(21, \left\lceil \sqrt{M \tfrac{D_y}{D_x}} \right\rceil\right),$$
$$\lambda_y = \max\left(21, \left\lceil \sqrt{M \tfrac{D_x}{D_y}} \right\rceil\right).$$

If we choose primes $p \in (\lambda_x..2\lambda_x]$ and $q \in (\lambda_y..2\lambda_y]$ independently and uniformly at random, then the probability that τ collides in $f(z^p, z^q)$ is less than μ.

Proof. Let $\tau = c_i x^{u_i} y^{v_i}$ for some $i \in [t]$. We will say a pair (p, q) is *bad* if τ collides with any other term in $f(z^p, z^q)$. Let $\mathcal{B}$ denote the set of bad pairs of primes (p, q) with $p \in (\lambda_x..2\lambda_x]$ and $q \in (\lambda_y..2\lambda_y]$. A collision with the ith term τ occurs when

$$u_i p + v_i q = u_j p + v_j q \tag{5.9}$$

for some $j \neq i$, $1 \leq j \leq t$. By reducing (5.9) modulo p and q, we see this may only occur if $p \mid (v_i - v_j)$ and $q \mid (u_i - u_j)$. Since $i \neq j$, it cannot be the case that both $(u_i - u_j)$ and $(v_i - v_j)$ are zero. Furthermore, if one $(u_i - u_j)$ or $(v_i - v_j)$ is zero, then the ith and jth term cannot collide. Thus, all collisions will occur at indices $j \in J$, where

$$J = \{j \in [t] : (u_i - u_j)(v_i - v_j) \neq 0\}.$$

For each prime $p \in (\lambda_x..2\lambda_x]$, define the subset of possible collision indices j as $J_p = \{j \in J : p \mid (v_i - v_j)\}$. We have that

$$\prod_{p \in (\lambda_x..2\lambda_x]} p^{\#J_p} \quad \text{divides} \quad \prod_{j \in J} (u_i - u_j) \leq D^{(T-1)},$$

where the product on the left-hand side is taken over all primes $p \in (\lambda_x..2\lambda_x]$. As the right-hand side product is nonzero by definition of J, it follows that if J_p is nonempty for some p, then

$$\sum_{p \in (\lambda_x..2\lambda_x]} \#J_p < (T-1) \ln D_y \ln^{-1} \lambda_x.$$

For each prime p, let $\mathcal{Q}_p$ be the number of primes $q \in (\lambda_y..2\lambda_y]$ such that (p, q) is bad. We have that

$$\prod_{q \in \mathcal{Q}_p} q \quad \text{divides} \quad \prod_{j \in J_p} (u_i - u_j) \leq D_x^{\#J_p}.$$

It follows that if $\#\mathcal{Q}_p$ is nonempty, then $\#\mathcal{Q}_p < \#J_p \ln D_x \ln^{-1} \lambda_y$, such that if $\mathcal{Q}_p$ is nonempty for any prime $p \in (\lambda_x..2\lambda_x]$, then

$$\sum_p \#\mathcal{Q}_p < \frac{\ln D_x}{\ln \lambda_y} \sum_p \#J_p < \frac{(T-1) \ln D_x \ln D_y}{\ln \lambda_x \ln \lambda_y}, \tag{5.10}$$

The left-hand side of (5.10) is precisely $\#\mathcal{B}$. Note that $M \leq \lambda_x \lambda_y$, such that by Lemma 2.4.4, the number of pairs of primes (p, q) with $p \in (\lambda_x..2\lambda_x]$ and $q \in (\lambda_y..2\lambda_y]$ is at least

$$\frac{9\lambda_x \lambda_y}{25 \ln \lambda_x \ln \lambda_y} \geq \frac{9M}{25 \ln \lambda_x \ln \lambda_y} = \frac{(T-1) \ln D_x \ln D_y}{\ln \lambda_x \ln \lambda_y} \mu^{-1} > \#\mathcal{B} \cdot \mu^{-1},$$

completing the proof. □

Corollary 5.3.3. *Let $f \in \mathsf{R}[x, y]$ with partial degress at most D_x, D_y and at most T nonzero terms. Then for any $\mu \in (0, 1)$ and primes p, q chosen at random as in Lemma 5.3.2, the substitution polynomial $g(z) = f(z^p, z^q)$ has degree at most*

$$\mathcal{O}\left(D_x + D_y + \sqrt{T D_x D_y \log(D_x) \log(D_y)}\right)$$

In comparison, the degree of a standard Kronecker substitution is $\mathcal{O}(D_x D_y)$. Because we can naively take T to be at most $D_x D_y$, the randomized substitution will never be more than a logarithmic factor greater than $D_x D_y$. When f is known to be sparse, i.e., when $T \ll D_x D_y$, we can obtain a substitution of appreciably reduced degree.

5.3.2 Multivariate substitutions

The analysis of the previous section fails in the n-variate case, where $n > 2$, as colliding terms may have exponents differing in two or more variables. As a result we do not have the same divisibility conditions as we used in the proof of Lemma 5.3.2. We take a different approach, choosing random integers as substitution vector components, not necessarily prime. This approach may apply to the bivariate case as well, and may be advantageous over the strategy of the previous section in the case that $T \ll \log(D_x D_y)$.

In this section, we write $f \in \mathsf{R}[x_1, \ldots, x_n]$ as

$$f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i},$$

where each $\boldsymbol{e}_i \in \mathbb{Z}^n$ is distinct, and $n \geq 2$. We let $D \geq \max_{i=1}^{n} \deg_{x_i}(f)$ and $T \geq t$ as usual.

Lemma 5.3.4. *Let $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i} \in \mathsf{R}[x_1, \ldots, x_n]$ with at most T nonzero terms, and $\tau = c_i \boldsymbol{x}^{\boldsymbol{e}_i}$ the ith term of f for some fixed $i \in [t]$. Fix $\mu \in (0, 1)$, let $\gamma > (T-1)/\mu$ be prime, and choose $\boldsymbol{s} \in \mathbb{Z}_\gamma^n$ uniformly at random. Then $\boldsymbol{e}_i^\top \boldsymbol{s} \not\equiv \boldsymbol{e}_j^\top \boldsymbol{s} \pmod{\gamma}$ for some $j \in [t] \setminus \{i\}$ with probability exceeding $1 - \mu$. In particular, τ collides in $f(z^{\boldsymbol{s}})$ with probability less than μ.*

Proof. Identify $[0..\gamma)$ with $\mathbb{Z}_\gamma$, and let $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_\gamma^n$ be a randomly-chosen vector. Consider the jth term of f, $j \in [t] \setminus \{i\}$. Let $\boldsymbol{u}_j = \boldsymbol{e}_i - \boldsymbol{e}_j \in \mathbb{Z}^n$. Observe that $\boldsymbol{u}_j \neq \boldsymbol{0}$, and that the ith and jth terms of f collide in $f(z^{s_1}, \ldots, z^{s_n})$ if and only if $\boldsymbol{u}_j^\top \boldsymbol{s} = 0$.

Let $\ell \geq 0$ be the largest integer such that γ^ℓ divides every entry in $\boldsymbol{u}_j$, and write $\boldsymbol{v}_j = \boldsymbol{d}_j \gamma^{-\ell}$. We have that $\boldsymbol{v}_j \in \mathbb{Z}^n$ and $\boldsymbol{u}_j \bmod \gamma \neq \boldsymbol{0}$. Furthermore, $\boldsymbol{u}_j^\top \boldsymbol{s} = 0$ if and only if $\boldsymbol{d}_j'^\top \boldsymbol{s} = 0$.

Now, if $\boldsymbol{v}_j^\top \boldsymbol{s} = 0$, then this also holds modulo γ, such that $\boldsymbol{s}$ lies in the $(n-1)$-dimensional null space of $\boldsymbol{v}_j^\top \bmod \gamma$. Denote this null space by $W \subseteq \mathbb{Z}_\gamma^n$. The probability that $\boldsymbol{s} \in W$ is γ^{-1}, such that the probability that terms i and j collide is at most γ^{-1} as well.

Taking the union bound over the $t-1$ indices $j \in [t] \setminus \{i\}$, we have that the probability that the ith term of f collides with any other term of f is at most $(t-1)\gamma^{-1} \leq (T-1)\gamma^{-1}$, which less than μ by the definition of γ. □

By Lemma 2.4.4, there exists a prime γ where $(T-1)/\mu < \gamma \leq \max(21, 2\lceil (T-1)/\mu \rceil) \in \mathcal{O}(T\mu^{-1})$. In the following Corollary we give the resulting degrees of the univariate polynomials:

Corollary 5.3.5. *Let $f \in \mathsf{R}[x_1, \ldots, x_n]$ with partial degrees at most D and at most T nonzero terms. For integers $s_1, \ldots, s_n$ chosen randomly as in Lemma 5.3.4, the polynomial $g(z) = f(z^{s_1}, \ldots, z^{s_n})$ has degree at most $\mathcal{O}(nDT\mu^{-1})$.*

5.4 Multivariate iterative sparse interpolation

We give an algorithm for the interpolation of a multivariate polynomial over an arbitrary commutative ring with identity. We exclude an analysis of the technique in the strictly-bivariate case, as the improvement over Kronecker substitution is less pronounced. The method described is a multivariate analogue of two-prime interpolation described in Section 3.4. As with that algorithm, we will maintain a sparse polynomial f_{sparse} such that $\#(f - f_{\text{sparse}})$ decreases by a factor of $1/2$.

We choose a substitution vector $\boldsymbol{s}$ such that *most* of the terms of f do not collide with respect to $\boldsymbol{s}$, and then construct a set of images corresponding to substitution vectors of the form $\boldsymbol{s} + \boldsymbol{r}_i$ for which we can identify terms in the images $f(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ with terms in the image $f(z^{\boldsymbol{s}})$. In this fashion we will be able to construct the terms of f that appear as singletons in $f(z^{\boldsymbol{s}})$, plus potentially additional deceptive terms constructed from information from collisions.

Similar to the selection of a $5/8$-support prime in two-prime interpolation, we will look for a substitution vector for which at most a proportion $3/8$ of the terms of g collide. We let a **$\boldsymbol{\sigma}$-support vector** for g denote a substitution vector $\boldsymbol{s}$ such that a proportion of at most σ terms of f are in collisions in the image $g(z^{\boldsymbol{s}})$.

As with two-prime interpolation, we would like a means of selecting an image with minimally many terms of g appearing in collisions. we show that a good candidate is the choice of $\boldsymbol{s}$ for which $g(z^{\boldsymbol{s}})$ has maximally many terms. We define $\mathcal{C}_g(\boldsymbol{s})$ to be the number of terms of g that are in collisions for a choice of substitution vector $\boldsymbol{s}$. By a completely analogous argument to that of Lemma 3.4.5 and Corollary 3.4.6, we claim the following lemma:

Lemma 5.4.1. *Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be substitution vectors such that $\#g(z^{\boldsymbol{u}}) \geq \#g(z^{\boldsymbol{v}})$. Then $\mathcal{C}_g(\boldsymbol{u}) \leq 2\mathcal{C}_g(\boldsymbol{v})$.*

From Lemma 5.4.1 we obtain the following proposition:

Proposition 5.4.2. *Let $g \in \mathsf{R}[\boldsymbol{x}]$ be n-variate and T-sparse. Let γ be a prime exceeding $16T$ and $\ell \geq \lceil \frac{1}{32} \ln(T\mu^{-1}) \rceil$. Choose $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell \in \mathbb{Z}_\gamma^n$ independently and uniformly at random. Then any of the $\boldsymbol{s}_i$ maximizing $\#g(z^{\boldsymbol{s}_i})$ is a $5/8$-support vector with probability at least $1 - \mu$.*

Proof. By Lemma 5.3.4, if we choose $\boldsymbol{s}$ uniformly at random from $\mathbb{Z}_\gamma^n$, then a fixed term τ of g collides with probability at least $1/16$. Then if we choose $\ell = \lceil \frac{1}{32} \ln(T\mu^{-1}) \rceil$ substitution vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell \in \mathbb{Z}_\gamma^n$ independently and uniformly at random, by Hoeffding's inequality each term collides for at most a proportion $3/16$ of the substitution vectors $\boldsymbol{s}_i$ with probability at least $1 - \mu/T$, such that by the union bound this occurs for all terms of g with probability at least $1 - \mu$. In which case, there must exist $\boldsymbol{s}_i$ for which a proportion of at most $3/16$ of the terms of g collide. □

Once we fix a substitution vector $\boldsymbol{s}$ and a substitution $g_0(z) \stackrel{\text{def}}{=} g(z^{\boldsymbol{s}})$, we pick a set of n-vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$, such that we can correlate terms from the substitutions $g_i = g(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ to terms in $g(z^{\boldsymbol{s}})$. We let $\boldsymbol{r}_i \in \mathbb{Z}_{\geq 0}^n$ be zero except for its ith component. We set $\boldsymbol{r}_i$[1] to some value w such that the formal terms of $g(z^{\boldsymbol{s}})$ with nonzero preimages have distinct exponents modulo w. Then we associate terms of $g(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ with single terms of $g(z^{\boldsymbol{s}})$.

If a term $c\boldsymbol{x}^{\boldsymbol{e}}$ of g appears as a singleton term cz^d in $g(z^{\boldsymbol{s}})$ then it will appear as cz^{d+we_i} in $g(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ for each $i \in [n]$. For each term cz^d in g_0 and each $i \in [n]$, we look for a unique term with degree d_i. If such terms exist for a given d and all $i \in [n]$, we construct a term $\tau = c\boldsymbol{x}^{\boldsymbol{e}}$, where $\boldsymbol{e} = (e_1, \ldots, e_n)$.

If a pair of terms $c_1\boldsymbol{x}^{\boldsymbol{e}_1}$ and $c_2\boldsymbol{x}^{\boldsymbol{e}_2}$ of g form a collision in $g(z^{\boldsymbol{s}})$ at degree d, then, for any $i \in [n]$ such that $e_{1i} \neq e_{2i}$, $g(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ will have two distinct terms of degrees $d_1 \neq d_2$ such that $d_1 \equiv d_2 \equiv d \pmod{w}$, namely $c_1 z^{d+we_{1i}}$ and $c_2 z^{d+we_{2i}}$. In this event we know that the degree-d term of g_0 is a not a singleton.

As few as three terms colliding may result in a deceptive term. One may produce an additional check to test if a constructed term τ is deceptive by testing that $\tau(z^{\boldsymbol{s}}) = z^d$, such that τ agrees with its supposed image in g_0. If we observe this then we know that the degree-d term of $g(z^{\boldsymbol{s}})$ is not a singleton; however, as we illustrate in the following example, this check is not robust.

Example 5.4.3. *Let $g(x_1, x_2, \ldots, x_6) = x_1x_2x_3x_4 + x_3x_4x_5x_6 - x_2x_3x_4x_5$. Suppose we choose a substitution vector $\boldsymbol{s} = (1, 1, 1, 1, 1, 1)$, such that the terms of g all collide at degree 4 in*

$$g_0 = g(z^{\boldsymbol{s}}) = g(z, z, z, z, z, z) = z^4.$$

[1] The author laments that "$\boldsymbol{e}_i$", the common notation for a standard basis element, is already in use.

Then, choosing $v = 1$, our approach will construct univariate images

$$g_1 = g(z^2, z, z, z, z, z) = z^5 + z^4 - z^4 = z^5, \quad g_2 = g(z, z^2, z, z, z, z) = z^5 + z^4 - z^5 = z^4,$$
$$g_3 = g(z, z, z^2, z, z, z) = z^5 + z^5 - z^5 = z^5, \quad g_4 = g(z, z, z, z^2, z, z) = z^5 + z^5 - z^5 = z^5,$$
$$g_5 = g(z, z, z, z, z^2, z) = z^4 + z^5 - z^5 = z^4, \quad g_6 = g(z, z, z, z, z, z^2) = z^4 + z^5 - z^4 = z^5.$$

We (erroneously) suppose that the z^4 term of g_0 is the image of a single term τ of g, and construct the partial exponents e_i of τ as $e_i = \deg(g_i) - \deg(g_0)$ for $i \in [6]$. This gives a deceptive term $\tau = x_1x_3x_4x_6$ that is not a term of g. Note that $\tau(z^{\boldsymbol{s}}) = z^4$, such that this deceptive term agrees with the image g_0.

Note that, in order for a deceptive term to occur, we need a collision of terms $\sigma \in g$ forming a multiple in the substitution $g(z^{\boldsymbol{s}})$, such that $\sigma(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ appears as a single, unique nonzero term for each $i \in [n]$. We argue this cannot happen in the bivariate case. If all of the terms of σ collide in $g(z^{\boldsymbol{s}})$, then the exponents of σ all lie in some translation of the nullspace of $\boldsymbol{s}^\top$. Moreover, for each $i \in [n]$, in order for $\sigma(z^{\boldsymbol{s}+\boldsymbol{r}_i})$ to appear as a single term, there must be a collision of some terms of σ under that substitution. That is, σ has a pair of terms $c_1\boldsymbol{x}^{\boldsymbol{e}_1}$ and $c_2\boldsymbol{x}^{\boldsymbol{e}_2}$ such that $\boldsymbol{e}_1 - \boldsymbol{e}_2$ is in the nullspace of $(\boldsymbol{s}+\boldsymbol{r}_i)^\top$, and hence $\boldsymbol{r}_i$, for each $i \in [2]$. As $\boldsymbol{e}_1 - \boldsymbol{e}_2$ is in the nullspace of $\boldsymbol{s}$ as well as $\boldsymbol{r}_i$, and $\boldsymbol{e}_1 = \boldsymbol{e}_2$, we must have that $\boldsymbol{s}$ and $\boldsymbol{r}_1$ have the same nullspace, and that $\boldsymbol{s}$ and $\boldsymbol{r}_1$ differ by a nonzero constant factor. By the same argument we get that $\boldsymbol{s}$ is a constant multiple of $\boldsymbol{r}_2$, a contradiction. We were unable to determine if a deceptive term may occur in the n-variate case, for $n \in [3..5]$.

We can take w, i.e., the amount we add to the i-th component of $\boldsymbol{s}$ in the image g_i, to be $B + 1$ for a degree bound $B \geq \deg(g(z^{\boldsymbol{s}}))$. The substitution g_0 will have degree at most $B = nD(\gamma - 1) \in \mathcal{O}(nDT)$. We can also take v to be a prime such that the formal terms of $g(z^{\boldsymbol{s}})$ with a nonzero preimage remain distinct modulo $(z^w - 1)$. We will call such terms of $g(z^{\boldsymbol{s}})$ its **significant formal terms**. Per the proof of Lemma 3.3.6, as there are at most T such terms, we can probabilistically select a "good prime" $w \in \mathcal{O}(T^2 \log B) = \mathcal{O}(T^2 \log(nDT))$. One may choose whichever possibility for w is smaller, which depends on the relative sparsity of g.

5.4.1 Interpolation

We describe our interpolation procedure in Algorithm 16. We state its correctness and cost as a Theorem.

Theorem 5.4.4. *Algorithm 16 interpolates a black-box polynomial f with probability at least $2/3$.*

- *$\mathcal{O}(\log^2 T)$ queries of degree $\mathcal{O}(nTD)$;*
- *$\mathcal{O}(n \log T)$ queries of degree $\mathcal{O}(nDT + DT \min(nD, T \log(nTD)))$.*

Algorithm 16: Multivariate iterative sparse interpolation

Input: $f \in \mathsf{R}[\boldsymbol{x}]$, an n-variate black-box polynomial; $D \geq \max_{i=1}^{n} \deg_{x_i}(f)$; $T \geq \#f$.
Output: With probability at least $2/3$, a sparse representation of f.

$T_g \leftarrow T$;
$f_{\text{sparse}} \leftarrow 0 \in \mathsf{R}[x_1, \ldots, x_n]$, a sparse polynomial;
$\mu \leftarrow 1/(3\lceil \log T + 1 \rceil)$;
while $T_g > 0$ **do**
  // Construct $g_0 = g(z^{\boldsymbol{s}})$ for a $5/8$-support substitution vector $\boldsymbol{s}$
  $\gamma \leftarrow$ a prime from $(16T_g..32T_g]$;
  $\mathcal{S} \leftarrow \ell = \lceil \frac{1}{32} \ln(2T_g\mu^{-1}) \rceil$ vectors $\boldsymbol{s} \in \mathbb{Z}_{\gamma}^{n}$ chosen uniformly at random;
  $g_0 \leftarrow (f - f_{\text{sparse}})(z^{\boldsymbol{s}})$, where $\boldsymbol{s} = \arg\max_{\boldsymbol{s} \in \mathcal{S}} \#(f - f_{\text{sparse}})(z^{\boldsymbol{s}})$;
  // Construct substitutions g_i, for $i \in [n]$
  $B \leftarrow nD(\gamma - 1)$;
  $w \leftarrow B + 1$;
  $\lambda \leftarrow \max(21, \lceil T(T-1)\ln B \rceil)$;
  if $\lambda \ll B$ **then**
    Choose $m = \lceil \log(2\mu^{-1}) \rceil$ primes $p_1, \ldots, p_m \in \mathcal{P}_{(\lambda..2\lambda]}$ at random;
    if $\#(g \bmod p_i) = \#g$ *for some* p_i **then** $w \leftarrow p_i$;
    else return `fail`;
  for $i \leftarrow 1$ **to** n **do**
    $\boldsymbol{r}_i \leftarrow i$th column of $w\boldsymbol{I}_n$, where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix;
    $g_i \leftarrow (f - f_{\text{sparse}})(z^{\boldsymbol{s}+\boldsymbol{r}_i})$;
  // Construct terms and update f_{sparse} and T_g
  for *each term* cx^d *of* g_0 **do**
    if *for each* $i \in [n]$, g_i *has a unique nonzero term of degree* $d_i \equiv d \pmod{v}$ **then**
      $\boldsymbol{e} \leftarrow w^{-1}(d_1 - d, d_2 - d, \ldots, d_n - d)$;
      if $\boldsymbol{e}^{\top}\boldsymbol{s} \neq d$ **then continue**;
      $f_{\text{sparse}} \leftarrow f_{\text{sparse}} + c\boldsymbol{x}^{\boldsymbol{e}}$;
  $T_g \leftarrow \lfloor T_g/2 \rfloor$;
  if $\#f_{\text{sparse}} > T_g + T$ **then return** `fail`;
return f_{sparse}

Proof. The algorithm has at most $\lceil \log T + 1 \rceil$ iterations of the outer while loop. Thus it suffices that each iteration succeeds with probability at least $1 - \mu$, where $\mu = 1/(3\lceil \log T + 1 \rceil)$. The

probabilistic steps of a single iteration of the while loop are the selection of a $5/8$ substitution vector $\boldsymbol{s}$, and (possibly), the selection of a good prime p_i for $g_0 = (f - f_{\text{sparse}})(z^{\boldsymbol{s}})$.

By Proposition 5.4.2, $\boldsymbol{s}$ is a $5/8$-support substitution vector with probability at least $1 - \mu/2$. The total degree of g_0 is at most $nD(\gamma - 1) = B$. By Lemma 3.3.6, a prime chosen at random from $(\lambda..2\lambda]$ is a good prime for g_0 with probability at least $1/2$, such that the probability that $m = \lceil \log(2\mu^{-1}) \rceil$ primes $p_1, \ldots, p_m \in (\lambda..2\lambda]$ are all not good primes for g_0 is $2^{-m} \leq \mu/2$. By the union bound an iteration succeeds with probability at least $1 - \mu$.

We now analyze the cost for a single iteration of the outer while loop. The selection of a $5/8$-support substitution vector $\boldsymbol{s}$ requires $\ell \in \mathcal{O}(\log T)$ calls to an interpolation procedure to compute images $(f - f_{\text{sparse}})(z^{\boldsymbol{s}})$ of degree at most $nD(\gamma - 1) \in \mathcal{O}(nTD)$. The n shifted substitutions $g_i, i \in [n]$ are T-sparse with degree at most

$$\begin{aligned} nDT + Dw = \mathcal{O}(nDT + D\min(B, \lambda)) &= \mathcal{O}(nDT + D\min(nDT, T^2\log(nDT))) \\ &= \mathcal{O}(nDT + DT\min(nD, T\log(nDT))). \end{aligned}$$

Accounting for all $\mathcal{O}(\log T)$ iterations gives the number of queries. □

We remark that the additional bit-cost due to selecting primes does not dominate the cost of constructing the univariate images. The selection of primes $p_1, \ldots, p_m \in \mathcal{P}_{(\lambda..2\lambda]}$, can be done with probability, say, $1 - \mu/100$ via GetPrimes with cost

$$\widetilde{\mathcal{O}}(m \cdot \text{polylog}(\lambda)\log(\mu^{-1})) \subseteq \widetilde{\mathcal{O}}(\text{polylog}(T\log\log(nD))),$$

which is less than the bit sizes of the sparse representations of the univariate images. The cost of identifying terms in the images g_i, $i \in [n]$, with terms in g_0 can be done via a dictionary with cost softly-linear in the bit size of $g_0, \ldots, g_n$.

We also note that one could instead of choosing v to be a prime for which a constant proportion of the significant terms of g_0 do not collide modulo $(z^v - 1)$. This would allow us to choose $w \in \mathcal{O}(T\log B) = \mathcal{O}(T\log(nDT))$. This would require a lengthier analysis.

5.5 Multivariate diversification

Multivariate iterative sparse interpolation allows us to collect terms from multiple substitutions by allowing for substitutions of larger degree. We can forego this by way of the multivariate analogue of diversification. We note that the maps $f \to f(z^{\boldsymbol{s}})$ are sparsity-preserving maps acting on the multivariate ring $\mathsf{R}[\boldsymbol{x}]$. We remind the reader of a notion of a *diversifying set*, now defined in a multivariate setting.

Definition 5.5.1. *Let $\Phi_1, \Phi_2, \ldots, \Phi_\ell : \mathsf{R}[\boldsymbol{x}] \to \mathsf{R}[\boldsymbol{y}]$ be sparsity-preserving maps. Define $f_i = f(\boldsymbol{z}^{\boldsymbol{s}_i})$, for $i \in [\ell]$. We say $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ form a* **diversifying set** *for f with images $f_i = \Phi_i(f), i \in [\ell]$ if, for any pair of formal terms $\tau_1 = c_1 \boldsymbol{x}^{\boldsymbol{e}_1}$ of f_i and $\tau_2 = c_2 \boldsymbol{x}^{\boldsymbol{e}_2}$ of f_j with respective nonzero preimages σ_1 and σ_2, $\sigma_1 \neq \sigma_2$, then there exists $k \in [m]$, such that $\sigma_1(\boldsymbol{a}_k) \neq \sigma_2(\boldsymbol{a}_k)$.*

As in the univariate case, a diversifying set for f and $f_1, \ldots, f_\ell$ will allow us to collect terms from $f_1, \ldots, f_\ell$ according to their preimage. For $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathsf{R}^n$, let

$$f_{ij} \stackrel{\text{def}}{=} f(\boldsymbol{a}_j z^{\boldsymbol{s}_i}) = f(a_{j1} z^{s_{i1}}, \ldots, a_{jn} z^{s_{in}}),$$

A diversifying set $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ is precisely a set that builds a one-to-one correspondence between nonzero preimages and vectorized coefficients. Consider the vector polynomial $\boldsymbol{f}$ given by

$$\boldsymbol{f}(\boldsymbol{x}) \stackrel{\text{def}}{=} (f(\boldsymbol{a}_1 \boldsymbol{x}), \ldots, f(\boldsymbol{a}_m \boldsymbol{x})) \in \mathsf{R}^m[\boldsymbol{x}], \tag{5.11}$$

and may write $\boldsymbol{f} = f(\boldsymbol{A}\boldsymbol{x})$, where the ith column of $\boldsymbol{A} \in \mathsf{R}^{n \times m}$ is given by $\boldsymbol{a}_i$. We also define the images

$$\boldsymbol{f}_i \stackrel{\text{def}}{=} \boldsymbol{f}(\boldsymbol{A} z^{\boldsymbol{s}_i}) = (f_{i1}, \ldots, f_{i\ell}) \in \mathsf{R}^n[z], \quad i \in [\ell].$$

If $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ forms a diversifying set for f with substitutions $f_1, \ldots, f_\ell$, then any pair of terms from the substitutions $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell$ that share a common nonzero vector coefficient $\boldsymbol{c}$ must have the same preimage.

We associate with each nonzero term $cx^e \in \boldsymbol{f}_i$ a key $\boldsymbol{c} \in \mathsf{R}^m$, where c_i is the coefficient of the degree-e term of $f_{ij} = f(\boldsymbol{a}_j z^{\boldsymbol{s}_i})$. By the definition of a diversifying set, collecting terms according to these keys will group the terms according to their preimage. The following Lemma allows us to probabilistically construct diversifying sets:

Lemma 5.5.2. *Let $\Phi_1, \ldots, \Phi_\ell : \mathsf{R}[\boldsymbol{x}] \mapsto \mathsf{R}[\boldsymbol{y}]$ be sparsity-preserving maps, $f \in \mathsf{R}[\boldsymbol{x}]$, $D^* \geq \deg(f)$, $T \geq \#f$ and $f_i = \Phi_i(f)$ for $i \in [\ell]$. Let $\mathcal{S} \subset \mathsf{R}$ be a finite regular-difference set of cardinality at least ρD^*, where $\rho > 1$. Let*

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

Choose $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ independently and uniformly at random from $\mathcal{S}^n$. Then $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ forms a diversifying set for f with $f_1, \ldots, f_\ell$ with probability at least $1 - \mu$.

We exclude a proof, but note that it follows essentially in the same way as the proof of Lemma 3.5.4. The only difference is that now we consider total degree bound by D^*, as opposed to univariate degrees bound by D. The DLSZ Lemma still applies. In the case that f has partial degrees bound by D, we can take $D^* = nD$, giving us this immediate Corollary in the case f is an extended black-box polynomial over $\mathbb{F}_q$.

Corollary 5.5.3. *Let $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ with partial degrees at most D and at most T terms, with substitutions $f_1, \ldots, f_\ell$, and $\mu \in (0,1)$, and fix a set of images $f_1, \ldots, f_\ell$ Let $u = \lceil \log_q(2nD+1) \rceil$, and let $m = \lceil 2\log T + 2\log(1+\ell/2) + \log(\mu^{-1}) - 1 \rceil$. Choose $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{F}_{q^u}$ independently and uniformly at random. Then $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ forms a diversifying set of f with $f_1, \ldots, f_\ell$ with probability at least $1-\mu$.*

5.6 Bivariate and multivariate majority-rule sparse interpolation

In this section we show how one can interpolate an extended black-box polynomial f using randomized substitutions with diversification. As with univariate majority-vote interpolation, we will construct all the terms of f in one batch. We let $D^* = nD$ be a bound on the total degree of f. We will consider the case that $f \in \mathbb{F}_q[\boldsymbol{x}]$, but the algorithms generalize to f over ring containing a regular-difference set of cardinality at least $2D^*$ (or, more generally, $(1+\epsilon)D^*$ for a constant $\epsilon > 0$).

If $T = 1$, then we simply perform n substitutions

$$f(z,1,\ldots,1), \quad f(1,z,1,\ldots,1), \quad \ldots, \quad f(1,\ldots,1,z),$$

each of which reveals the single term and its exponent in one of the variables. No randomization here is necessary. For the remainder of the Chapter we assume $T \geq 2$.

5.6.1 Choosing substitutions and a diversifying set

We first select ℓ randomized Kronecker substitutions $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell$. We will require that the $\boldsymbol{s}_i$ are chosen in such a way that, with high probability, every term of f avoids collision for at least half of the substitutions $\boldsymbol{s}_i$. To achieve this we first randomly select substitutions in a manner such that any fixed term of f avoids collision for a fixed substitution $\boldsymbol{s}$ with probability exceeding $3/4$.

In order to achieve this in the bivariate case, we choose primes p and q according to Lemma 5.3.2, and set a substitution vector $\boldsymbol{s} = (p, q)$. That is, we set

$$\lambda_x = \max\left(21, \left\lceil \tfrac{10}{9}\sqrt{(T-1)^{D_y/D_x} \ln D_x \ln D_y} \right\rceil\right),$$
$$\lambda_y = \max\left(21, \left\lceil \tfrac{10}{9}\sqrt{(T-1)^{D_x/D_y} \ln D_x \ln D_y} \right\rceil\right).$$

and randomly choose primes $p \in (\lambda_x .. 2\lambda_x]$ and $q \in (\lambda_y .. 2\lambda_y]$ independently and uniformly at random. In the general n-variate case we choose $\boldsymbol{s} \in [0..\gamma)^n$, where we follow Lemma 5.3.4 and take γ to be a prime where $\gamma > 4(T-1)$.

The following lemma shows how many substitutions ℓ are required such that every term of f appears without collisions in at least half of the images.

Lemma 5.6.1. *Let $f \in \mathsf{R}[x_1, \ldots, x_n]$ be T-sparse. Set*

$$\ell = \max(4n, \lceil 8 \ln(T\mu^{-1}) \rceil),$$

and choose ℓ vectors $\boldsymbol{s} \in \mathbb{Z}^n_{\geq 0}$ such that, for any single $\boldsymbol{s}$ and for any fixed term τ of f, the probability that τ collides with another is less than $1/4$. Then, with probabilty at least $1 - \mu$, every term of f collides with no others for at least $2n$ of the substitutions.

Proof. By Hoeffding's inequality (Thm. 2.3.4), the probability that any fixed term of f collides in a proportion of at least $1/2$ of the substitutions is less than $\exp(-\ell/8) \leq T^{-1}\mu$. Taking the union bound over the at most T terms of f completes the proof. □

We choose substitution vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell$ and construct the substitutions

$$f_i = f(z_i^{\boldsymbol{s}}) = f(z^{s_{i1}}, z^{s_{i2}}, \ldots, z^{s_{in}}), \quad i \in [\ell]. \tag{5.12}$$

By Lemma 5.5.2, if we take $r = \lceil \log_q(2nD) \rceil$ and $m = \lceil 2 \log T + 2 \log(1 + \ell/2) + 2 \rceil$ and choose $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{F}_{q^r}^m$ independently and uniformly at random, then $\mathcal{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$ forms a diversifying set with probability at least $7/8$.

If $\mathcal{A}$ forms a diversifying set, then distinct vector coefficients occurring in the images $\boldsymbol{f}_i$ given by (5.11) will each have distinct preimages. Then, by Lemma 5.6.1, any nonzero vector coefficients that appears in over half of the images $\boldsymbol{f}_i$ will have a preimage that is a distinct term of f.

5.6.2 Recovering the multivariate exponents

For each vector coeffcient $\boldsymbol{c}$ that appears in at least $\ell/2$ of the images $\boldsymbol{f}_i$, we attempt to find a set of n substitutions $f_i(\alpha)$ for which $\boldsymbol{f}_i$ contains the coefficient $\boldsymbol{c}$, and the corresponding substitution vectors are linearly independent. In the bivariate case this is not difficult as any 2×2 linear system formed by two substitution vectors will have nonzero determinant since in this case the substitution vectors are distinct ordered pairs of primes.

The general multivariate case is more invovled, as a choice of n substitution vectors may not always be linearly independent. For this case we suppose there are at least $2n$ vectors $\boldsymbol{s}_i, i \in \mathcal{I}_{\boldsymbol{c}}$, where $\mathcal{I}_{\boldsymbol{c}} \subset [\ell]$ which give vector images $\boldsymbol{f}_i$ each containing the vector coefficient $\boldsymbol{c}$. In order to construct a unique exponent, we require that the matrix whose rows are given by $\boldsymbol{s}_i, i \in \mathcal{I}_{\boldsymbol{c}}$, has full rank. This requires some care because this probability is conditional on the fact that some fixed term of f does not collide for each of the $\boldsymbol{s}_i$.

Lemma 5.6.2. *Let $f \in \mathsf{R}[x_1, \ldots, x_n]$ and λ be a prime number, and suppose that a term of f avoids collision in an image $f(\boldsymbol{x}^{\boldsymbol{s}})$, for a randomly-chosen $\boldsymbol{s} \in [0..\lambda)^n$, with probabilty at least $3/4$. Let $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{2n}$ be row vectors chosen independently and uniformly at random from $[0..\lambda)^n$. Given that a fixed term $\tau \in f$ avoids collision in the images $f(z^{\boldsymbol{r}_i})$, $i \in [2n]$, then for the matrix*

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_{2n} \end{bmatrix}^\top \in \mathbb{Z}_\lambda^{2n \times n}, \tag{5.13}$$

$\boldsymbol{S} \bmod \lambda$ has rank less than n with probability at most $(\frac{9}{16}\lambda)^{-n}$.

Note that if $\boldsymbol{S} \bmod \lambda$ has full rank, then $\boldsymbol{S}$ is as well.

Proof. If $\boldsymbol{S}$ has rank less than n, then $\boldsymbol{S} \bmod \lambda$ also has rank less than n. We identify $\{0, 1, \ldots, \lambda - 1\}$ with $\mathbb{Z}_\lambda$ and consider the $\boldsymbol{s}_i$ and $\boldsymbol{S}$ as being over $\mathbb{Z}_\lambda$.

Suppose then that $\boldsymbol{S}$ has rank less than n. Then $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{2n}$ all lie in some $(n-1)$-dimension subspace $W \subseteq \mathbb{Z}_\lambda^n$, of cardinality λ^{n-1}. Thus W may be specified by a nonzero vector spanning its orthogonal space, unique up to a scalar multiple. It follows that the number of such possible subspaces W is less than λ^n, and so the number of possible $2n$-tuples comprised of substitution vectors that do not span $\mathbb{Z}_\lambda^n$ is at most

$$\lambda^n (\lambda^{n-1})^{2n} = \lambda^{2n^2 - n}.$$

Meanwhile, there are λ^{2n^2} possible $2n$-tuples of substitution vectors from λ^n, and thus the probability that such a $2n$-tuple does not span W is a most $\lambda^{2n^2 - (2n^2 - n)} = \lambda^{-n}$. Furthermore, by the hypothesis, the probability that a term of f avoids collision for every substitution $f(z^{\boldsymbol{s}_i})$ is at least $(3/4)^{2n}$. Thus the conditional probability that $\boldsymbol{S}$ is not full rank, given that a fixed term of f avoids collision for each $\boldsymbol{s}_i$ is at most $\lambda^{-n}(3/4)^{-2n} = (\frac{9}{16}\lambda)^{-n}$. □

From this we show that, with high probability, *every* term $\tau \in f$ admits such a full rank matrix $\boldsymbol{S}$ of substitution vectors in whose corresponding substitution τ does not collide.

Corollary 5.6.3. *Let $\lambda \geq 4T$ and $f \in \mathsf{R}[x_1, \ldots, x_n]$ be T-sparse, where $T \geq 2$ and $n \geq 3$. Suppose that any fixed term $\tau \in f$ avoids collision for a randomly selected substitution vector $\boldsymbol{s} \in [0..\lambda)^n$ with probability at least $3/4$. Choose $s_1, \ldots, s_\ell \in [0..\lambda)^n$ independently and uniformly at random.*

If every term $\tau \in f$ avoids collision for at least $2n$ substitution vectors $\boldsymbol{s}_{\tau 1}, \ldots, \boldsymbol{s}_{\tau m} \in \{\boldsymbol{s}_i : i \in [\ell]\}$, then with probability at least $713/729$, $\boldsymbol{S}_\tau$ has full rank for every term $\tau \in f$, where $\boldsymbol{S}_\tau = [\boldsymbol{s}_{\tau 1} \cdots \boldsymbol{s}_{\tau 2n}]$.

Proof. By Lemma 5.6.2 the probability that a single term does not have a full-rank set of good substitution vectors is at most $(\frac{9}{16}\lambda)^{-n}$. By the union bound, and using that $\lambda \geq 4T \geq 8$, the probability that any of the at most $T \geq 2$ terms of f each have such a full-rank set is less than

$$T\left(\frac{16}{9\lambda}\right)^n \leq \frac{\lambda}{4}\left(\frac{16}{9\lambda}\right)^n = \frac{4}{9}\left(\frac{16}{9\lambda}\right)^{n-1} \leq \frac{4}{9}\left(\frac{2}{9}\right)^2 = \frac{16}{729}.$$ □

For every nonzero vector coefficient $\boldsymbol{c}$ occuring in any of the vector polynomials $\boldsymbol{f}_i$, we maintain a matrix $\boldsymbol{S}_{\boldsymbol{c}}$ whose rows are $\boldsymbol{s}_i^\top, i \in [\ell]$ such that $\boldsymbol{f}_i$ has some term of the form $\boldsymbol{c}x^{d_i}$. We may implicitly store $\boldsymbol{S}_{\boldsymbol{c}}$ as the list of indices i corresponding to the $\boldsymbol{s}_i$ that comprise the $\boldsymbol{s}_{\boldsymbol{c}j}$. For any vector coefficient $\boldsymbol{c}$ that results in a matrix $\boldsymbol{S}_{\boldsymbol{c}}$ of more than $2n$ rows, we randomly truncate $\boldsymbol{S}_{\boldsymbol{c}}$ to $2n$ rows by removing some at random, and then solve the $2n \times n$ overdetermined system

$$\boldsymbol{S}_{\boldsymbol{c}} \begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} d_{\boldsymbol{c},1} \\ \vdots \\ d_{\boldsymbol{c},2n} \end{bmatrix}, \quad \text{where } \boldsymbol{S}_{\boldsymbol{c}} = \begin{bmatrix} \boldsymbol{s}_{\boldsymbol{c},1}^\top \\ \vdots \\ \boldsymbol{s}_{\boldsymbol{c},2n}^\top \end{bmatrix},$$

and $d_{\boldsymbol{c},i}$ is the degree of the term with coefficient $\boldsymbol{c}$ appearing in the substitution $f(z^{\boldsymbol{s}_{\boldsymbol{c},i}})$.

Remark 5.6.4. *A fixed collision σ of terms of f may appear in $2n$ or more images f_i, however, it will not give a full-rank linear system. If a pair of terms $c_1\boldsymbol{x}^{\boldsymbol{e}_1}, c_2\boldsymbol{x}^{\boldsymbol{e}_2} \in f$ collided to form a multiple for substitution vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ forming a full-rank system, then we would have that*

$$\begin{bmatrix} \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_n \end{bmatrix}^\top (\boldsymbol{e}_1 - \boldsymbol{e}_2) = \boldsymbol{0},$$

such that $\boldsymbol{e}_1 = \boldsymbol{e}_2$.

We give pseudocode for the bivariate case with distinct partial degree bounds in Algorithm 17, and the general multivariate case with a total degree bound. The algorithms differ in the selection of the substitution vectors, and the construction of a full-rank linear system. Otherwise the algorithms are essentially the same.

Per the preceding analysis, the bivariate algorithm succeeds with probability at least $3/4$, and the univariate algorithm succeeds with probability at least $3/4 - 16/729 \geq 8/11$.

We now analyze the costs. To simplify the analysis in the bivariate case we consider the case when $D_x = D_y = D$ such that $\lambda_x = \lambda_y = \lambda$ for some constant λ. In the bivariate case we chose $\ell = \mathcal{O}(\log T)$ substitutions vectors with entries of magnitude at most $\mathcal{O}(\lambda) \subseteq \mathcal{O}(\sqrt{T}\log D)$, and a diversifying set of $m \in \mathcal{O}(\log T + \log \ell) = \mathcal{O}(\log T)$ vectors $\boldsymbol{a}_j \in \mathsf{R}$. The univariate degrees of the substitutions are, per Corollary 5.3.3 at most $\mathcal{O}(D \log D\sqrt{T})$. In other words, the query cost is $\mathcal{O}(\log^2 T)$ shifted queries of degree $\mathcal{O}(D \log D\sqrt{T})$.

The bit cost of generating $2\ell \in \mathcal{O}(\log T)$ primes $p_i, q_i \in \mathcal{O}(\lambda)$ via a Monte Carlo method (e.g., **GetPrimes**) is

$$\widetilde{\mathcal{O}}(\ell \log \lambda) \subset \widetilde{\mathcal{O}}(\log^2 T + \log T \log\log D).$$

We have to solve $2T$ 2×2 linear systems $\boldsymbol{S}_{\boldsymbol{c}}\boldsymbol{e} = \boldsymbol{d}$ where the entries of $\boldsymbol{S}_{\boldsymbol{c}}$ and $\boldsymbol{d}$ are $\widetilde{\mathcal{O}}(D\sqrt{T})$. Using the "grade-school" method of solving the at-most T linear systems gives a bit cost of $\widetilde{\mathcal{O}}(T\log(D\sqrt{T})) = \widetilde{\mathcal{O}}(T\log D)$. We can choose 2ℓ primes from $(\lambda..2\lambda]$ with probability at least, say, $99/100$, with a bit cost of $\widetilde{\mathcal{O}}(\ell \cdot \mathrm{polylog}(\lambda)) \in \widetilde{\mathcal{O}}(\mathrm{polylog}(T\log D))$. As the total bit size of the sparse representation of the substitutions f_{ij} is $\Omega(T\log D)$, the query cost must dominate the total cost of the algorithm. We state its cost as a Theorem.

Algorithm 17: Bivariate majority-rule interpolation

Input: $f \in \mathbb{F}_q[x,y]$, an extended black-box polynomial; $D_x \geq \deg_x(f)$; $D_y \geq \deg_y(f)$; $D^* = D_x D_y$; $T \geq \#f$.

Output: A sparse representation of f, with probability at least $2/3$.

// Choose substitution vectors

$\ell \leftarrow \lceil 8 \ln(8T) \rceil$;

$\lambda_x = \max\left(21, \left\lceil \frac{10}{9} \sqrt{(T-1)^{D_y/D_x} \ln D_x \ln D_y} \right\rceil\right)$;

$\lambda_y = \max\left(21, \left\lceil \frac{10}{9} \sqrt{(T-1)^{D_x/D_y} \ln D_x \ln D_y} \right\rceil\right)$;

Choose distinct primes $p_1, \ldots, p_\ell \in (\lambda_x..2\lambda_x]$ and $q_1, \ldots, q_\ell \in (\lambda_y..2\lambda_y]$ independently and uniformly at random;

for $i \leftarrow i$ **to** ℓ **do** $\boldsymbol{s}_i \leftarrow (p_i, q_i)$;

// Choose diversifications

$r \leftarrow \log_q(2nD)$;

$m \leftarrow \lceil 2\log(T/2) + 2\log(1+\ell/2) + 2 \rceil$;

Choose $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{F}_{q^r}^2$ independently and uniformly at random;

// Construct substitutions

for $i \leftarrow 1$ **to** ℓ **do**

  Interpolate $f_i = f(z^{\boldsymbol{s}_i}) \in \mathbb{F}_{q^r}[z]$;

  for $j \leftarrow 1$ **to** m **do** Interpolate $f_{ij} = f(\boldsymbol{a}_j z^{\boldsymbol{s}_i}) \in \mathbb{F}_{q^r}[z]$;

  $\boldsymbol{f}_i \leftarrow (f_{i1}, \ldots, f_{im}) \in \mathbb{F}_{q^r}^m[x]$;

// Construct exponents via linear systems

$g \leftarrow 0 \in \mathbb{F}_q[\boldsymbol{x}]$, a sparse polynomial;

for *every coefficient* $\boldsymbol{c} \neq \boldsymbol{0}$ *appearing in at least* 2 *substitions* $\boldsymbol{f}_i \neq \boldsymbol{f}_j$, $i \neq j \in [\ell]$ **do**

  $\mathbf{S}_{\boldsymbol{c}} \leftarrow [\boldsymbol{s}_i \ \boldsymbol{s}_j]^\top \in \mathbb{Z}^{2\times 2}$;

  for $k \in \{i, j\}$ **do** $d_k \leftarrow$ degree of term with $\boldsymbol{c}$ coefficient in $\boldsymbol{f}_i$;

  $\boldsymbol{d} \leftarrow (d_i, d_k) \in \mathbb{Z}^2$;

  $\boldsymbol{e} \leftarrow$ solution to $\mathbf{S}_{\boldsymbol{c}}\boldsymbol{e} = \boldsymbol{d}$;

  $c \leftarrow$ coefficient of degree-d_1 term of f_i;

  $g \leftarrow g + c\boldsymbol{x}^{\boldsymbol{e}}$;

return g;

Algorithm 18: Multivariate majority-rule interpolation

Input: $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, an extended black-box polynomial; $D \geq \max_{i=1}^{n} \deg_{x_i}(f)$; $D^* = nD$; $T \geq \max(\#f, 2)$.

Output: A sparse representation of f, with probability at least $2/3$.

// Choose substitution vectors

$\ell \leftarrow \max(4n, \lceil 8 \ln(8T) \rceil)$;

$\lambda \leftarrow$ a prime in $(4T..8T]$;

Choose $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell \in [\lambda]^n$ independently and uniformly at random;

// Choose diversifications

$r \leftarrow \lceil \log_q 2nD \rceil$;

$m \leftarrow \lceil 2\log(T/2) + 2\log(1 + \ell/2) + 2 \rceil$;

Choose $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{F}_{q^r}^n$ independently and uniformly at random;

// Construct substitutions

for $i \leftarrow 1$ **to** ℓ **do**

  Interpolate $f_i = f(z^{\boldsymbol{s}_i}) \in \mathbb{F}_q[z]$;

  for $j \leftarrow 1$ **to** m **do** Interpolate $f_{ij} = f(\alpha_j z^{\boldsymbol{s}_i}) \in \mathbb{F}_{q^r}[z]$;

  $\boldsymbol{f}_i \leftarrow (f_{i1}, \ldots, f_{im}) \in \mathbb{F}_{q^r}^m[z]$;

// Construct exponents via linear systems

$g \leftarrow 0 \in \mathbb{F}_q[\boldsymbol{x}]$, a sparse polynomial;

for *every coefficient* $\boldsymbol{c} \neq \boldsymbol{0}$ *appearing in* $2n$ *substitutions* $f(z^{\boldsymbol{s}})$ *for vectors* $\boldsymbol{s} \in \{\boldsymbol{s}_{\boldsymbol{c},1}, \ldots, \boldsymbol{s}_{\boldsymbol{c},2n}\} \subseteq \{\boldsymbol{s}_i : i \in [\ell]\}$ **do**

  $\boldsymbol{S}_{\boldsymbol{c}} \leftarrow [\boldsymbol{s}_{\boldsymbol{c},1} \ \cdots \ \boldsymbol{s}_{\boldsymbol{c},2n}]^\top \in \mathbb{Z}^{2n \times n}$;

  for $i \leftarrow 1$ **to** $2n$ **do** $d_i \leftarrow$ degree of term with $\boldsymbol{c}$ coefficient in $\boldsymbol{f}_i$;

  $\boldsymbol{d} \leftarrow (d_1, \ldots, d_{2n}) \in \mathbb{Z}^{2n}$;

  Find solution $\boldsymbol{e}$ to $\boldsymbol{S}_{\boldsymbol{c}}\boldsymbol{e} = \boldsymbol{d}$ via a fast Monte Carlo version of the algorithm of Thm. 2.1.9;

  $c \leftarrow$ coefficient of degree-d_1 term of $f_{c,1}$;

  $g \leftarrow g + cx^{\boldsymbol{e}}$;

return g;

Theorem 5.6.5 (c.f. Algorithm 17)**.** *There exists a Monte Carlo algorithm that, given a bivariate extended black-box polynomial $f \in \mathbb{F}_q[x, y]$ and bounds $D \geq \max(\deg_x(f), \deg_y(f))$ and $T \geq \#f$ and, if $\#Z \geq 4D$, interpolates f and requires $\mathcal{O}(\log^2 T)$ $\mathbb{F}_{q^r}$-shifted queries of degree $\mathcal{O}(D \log D\sqrt{T})$, where $r = \lceil \log_q(2nD) \rceil$.*

In the multivariate case we choose $\ell = \mathcal{O}(n + \log T)$ substitutions with entries bounded by $\lambda \in \mathcal{O}(T)$, and $m \in \mathcal{O}(\log T + \log \ell)$ diversification vectors. The resulting univariate images are of degree $\mathcal{O}(nD\lambda) \subseteq \mathcal{O}(nDT)$. We note that the combined bit size of the sparse representations of images f_{ij} is $\widetilde{\mathcal{O}}(\ell m T n \log D) \in \widetilde{\mathcal{O}}(n^2 T \log D)$.

Here the cost of linear system solving may affect the soft-oh cost if we take n as a parameter. In order to achieve a Monte Carlo algorithm, it suffices to solve the at-most T $2n \times n$ linear systems $\mathcal{S}_c \boldsymbol{e} = \boldsymbol{d}$, each with probability at least $1 - \epsilon$, where $\epsilon < T/100$. We can do this, e.g., using the algorithm of Theorem 2.1.9, with cost

$$\widetilde{\mathcal{O}}(n^\omega T \log(TD) \log(\epsilon)) = \widetilde{\mathcal{O}}(n^\omega T \log D).$$

This exceeds the combined bit size of the sparse images f_{ij}. We summarize with the following Theorem:

Theorem 5.6.6. *[c.f. Algorithm 18] There exists a Monte Carlo algorithm that, given an n-variate extended black-box polynomial $f \in \mathbb{F}_q[\boldsymbol{x}]$ with partial degrees bounded by D, and at most T terms, and interpolates f with cost:*

- *$\mathcal{O}((n + \log T) \log T)$ $\mathbb{F}_{q^r}$-shifted queries of degree $\mathcal{O}(nDT)$, where $r = \lceil \log_q(2nD) \rceil$, and*
- *an additional $\mathcal{O}(n^\omega T \log D)$ bit operations due to linear system solving.*

In the next section we will show how to reduce the cost when f is highly multivariate, by choosing substitutions that form structured matrices.

5.7 Ultravariate majority-rule sparse interpolation

One caveat of the approach of the last section is that it potentially requires that we solve a unique linear system for every term of f. In this section we present an interpolation method for sparse, *ultravariate* (i.e. highly multivariate) extended black-box polynomials. This appears in [AGR15], and is joint work with Mark Giesbrecht and Daniel S. Roche. We also include herein an idea from Clement Pernet that further reduced the linear algebraic overhead by using structured linear algebra. It is a synthesis of ideas presented in Sections 3.5 and 5.3.

The techniques herein rely on diversification again. As such our algorithm can extend to polynomials over a R containing a sufficiently large regular-difference set. For the purposes of the description here we consider an extended black-box polynomial f over a finite field $\mathbb{F}_q$.

The idea is to work with images of the form

$$f(\boldsymbol{x})^{\bmod p} \stackrel{\text{def}}{=} f(\boldsymbol{x}) \bmod \langle x_1^p - 1, x_2^p - 1, \ldots, x_n^p - 1\rangle,$$

for a choice of primes p. For each choice of prime p, we will only construct *some* of the terms of f. We will choose substitutions $\boldsymbol{s} \in \mathbb{Z}_p^n$, and construct univariate images

$$f(z^{\boldsymbol{s}})^{\bmod p} = f(z^{s_1}, \ldots, z^{s_n}) \bmod (z^p - 1).$$

We do this for n vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$, which form an $n \times n$ matrix over $\mathbb{Z}_p$. Then, using diversification and collecting terms according to their coefficient, we are able to reconstruct $\boldsymbol{e} \bmod p$ for a significant proportion of the exponents $\boldsymbol{e} \in \operatorname{supp}(f)$ by solving a single $n \times n$ linear system over $\mathbb{Z}_p$. We do this for $\ell \in \mathcal{O}(\log D + \log T)$ primes p_i in a manner such that, with high probability and for every term $\tau \in f$, we can construct $\tau \bmod (\boldsymbol{x}^{p_i} - 1)$ for over half of the primes p_i. We then construct the exponents of f by way of Chinese remaindering. In this way, instead of solving T integer linear systems, we solve $\mathcal{O}(\log D + \log T)$ linear systems modulo a prime.

Now a term of f can collide in one of two ways. Two terms can collide for a choice of prime p if their componentwise difference modulo p is the zero vector, i.e., $\boldsymbol{x}^{\boldsymbol{e}_1}$ and $\boldsymbol{x}^{\boldsymbol{e}_2}$ collide if $(e_{1i} - e_{2i}) \bmod p = 0$ for all $i \in [n]$. We will call this type of collision a **prime collision**. If those terms are not in a prime collision, they can still collide for a choice of substitution vector $\boldsymbol{s} \in \mathbb{Z}_p^n$. I.e., we can have that $(\boldsymbol{e}_1 - \boldsymbol{e}_2) \bmod p \neq \boldsymbol{0}$, but $\boldsymbol{s}^\top(\boldsymbol{e}_1 - \boldsymbol{e}_2) \bmod p = 0$. We will call this type of collision a **substitution collision**.

5.7.1 Selecting primes and substitution vectors

In order to appropriately select primes, we give a multivariate generalization of Lemma 3.5.1. The proof follows very similarly.

Lemma 5.7.1. *Let $f \in \mathsf{R}[\boldsymbol{x}]$ be an n-variate polynomial with $t \leq T$ terms and partial degrees at most D, $\mu \in (0, 1)$, and*

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

Choose a prime p uniformly at random from $(\lambda..2\lambda]$. The probability that a fixed term $\tau \in f$ is not in an prime collision in $f(\boldsymbol{x})^{\bmod p}$ exceeds $1 - \mu$.

Proof. Write $f = \sum_{i=1}^{t} c_i \boldsymbol{x}^{\boldsymbol{e}_i}$, and, without loss of generality, suppose τ is the term $c_1 x^{\boldsymbol{e}_1}$. Let $\mathcal{B}$ comprise the set of "bad" primes $p \in (\lambda..2\lambda]$ such that τ is in a prime collision in the image

$f(\boldsymbol{x}) \bmod p$. If $c_1\boldsymbol{x}^{\boldsymbol{e}_i}$ and $c_i\boldsymbol{x}^{\boldsymbol{e}_i}$, collide in $f(\boldsymbol{x}) \bmod p$, then p divides $e_{1j} - e_{ij}$ for every $j \in [n]$. Furthermore, for every $i \in (1..t]$, there must exist some $j(i) \in [n]$ such that $e_{1,j(i)} \neq e_{i,j(i)}$. It follows that, for each $p \in \mathcal{B}$, p divides the nonzero integer $\prod_{i=2}^{t}(e_{1,j(i)} - e_{i,j(i)})$. Thus, if $\mathcal{B}$ is nonempty, we have

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

which gives us $\#\mathcal{B} \leq (T-1)\ln D \ln^{-1}\lambda$. By Lemma 2.4.4, the number of primes in $(\lambda..2\lambda]$ is at least $\frac{3}{5}\lambda\ln^{-1}\lambda \geq (T-1)\ln D\ln^{-1}\lambda\mu^{-1}$, completing the proof. □

Unlike in previous sections, we will choose substitution vectors $\boldsymbol{s}_{p1}, \ldots, \boldsymbol{s}_{pn}$ that form a Hankel matrix $\boldsymbol{S}$, such that the cost of the resulting $n \times n$ linear system solving over $\mathbb{Z}_p$ is reduced from $\widetilde{\mathcal{O}}(n^\omega)$ to $\widetilde{\mathcal{O}}(n)$. This idea is due to Clément Pernet, and deviates from the algorithm given in [AGR15]. The following Lemma, due to Pernet, shows that a randomly selected Hankel matrix will produce substitution collisions with bounded probability.

Lemma 5.7.2 (C. Pernet, personal correspondence)**.** *Let $f = \sum_{i=1}^{t} \in \mathsf{R}[x]$ with $t \leq T$ terms. Choose $s_1, \ldots, s_{2n} \in \mathbb{Z}_p$ independently and uniformly at random. Then a term τ of f avoids substitution collisions for*

$$\boldsymbol{s}_i = (s_{i+1}, \ldots, s_{i+n}) \in \mathbb{Z}_p^n, \tag{5.14}$$

for all $i \in [n]$ with probability at least $1 - n(T-1)/p$.

Proof. Consider a pair of terms $\tau_1 = c_1\boldsymbol{x}^{\boldsymbol{e}_1}$ and $\tau_2 = c_2\boldsymbol{x}^{\boldsymbol{e}_2}$. If τ_1 and τ_2 are in a substitution collision, then we have that $\boldsymbol{e}_1 - \boldsymbol{e}_2 \neq \boldsymbol{0} \bmod p$, such that there exists $j \in [n]$ such that $e_{1j} \neq e_{2j}$. Without loss of generality suppose $j = n$.

For each $i \in [n]$ and for a fixed choice of $s_i, \ldots, s_{i+n-1} \in \mathbb{Z}_p$, we have that $\boldsymbol{s}_i^\top \boldsymbol{e} = 0$ if and only if

$$s_{i+n} = \frac{\sum_{k=1}^{n-1} s_k(e_{1k} - e_{2k})}{e_{1n} - e_{2n}}, \tag{5.15}$$

such that there is a unique choice of s_{i+n} for which $\boldsymbol{s}_i^\top(\boldsymbol{e}_1 - \boldsymbol{e}_2) = 0$, and τ_1 and τ_2 collide with probability at least $1 - 1/p$. The lemma then follows from the union bound. □

Lemma 5.3.4 gives a means of bounding the probability that a term is in a substitution collision. We take $p > 10n(T-1)$, and $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n \in \mathbb{Z}_p^n$ are chosen independently and uniformly at random, then a fixed term of f will not collide in any substitution $f(z^{\boldsymbol{s}_i}), i \in [n]$, with probability at least $9/10$.

We will require, in addition, that $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n \in \mathbb{Z}_p^n$ form a nonsingular matrix $\boldsymbol{S} \in \mathbb{Z}_p^{n\times n}$. In a non-theoretical setting, we may just reselect $s_1, \ldots, s_{2n} \in \mathbb{Z}_p$, until we get a nonsingular

matrix; however, for the purposes of obtaining a deterministic runtime, our algorithm will merely disregard that particular prime p. To probabilistically guarantee that $\boldsymbol{S}$ is nonsingular we cite the following Theorem.

Theorem 5.7.3 (Corollary 2, [KL96])**.** *Let p be prime. Then for $s_1, \ldots, s_{2n} \in \mathbb{Z}_p$ chosen independently and uniformly at random, the probability that the Hankel matrix*

$$\boldsymbol{S}_p = \begin{bmatrix} s_1 & s_2 & \cdots & s_n \\ s_2 & s_3 & \cdots & s_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_n & s_{n+1} & \cdots & s_{2n} \end{bmatrix} \in \mathbb{Z}_p^{n \times n} \tag{5.16}$$

is nonsingular with probability $1 - 1/p$.

By Lemmata 5.3.4 and 5.7.2, Theorem 5.7.3, and the union bound we have the following Corollary.

Corollary 5.7.4. *Let*

$$\lambda \geq \lceil \max(21, \quad \tfrac{50}{3}(T-1)\ln D, \quad 10n(T-1)) \rceil. \tag{5.17}$$

Choose a prime p uniformly at random from $(\lambda..2\lambda]$, and $s_1, \ldots, s_{2n}$ independently and uniformly at random from $\mathbb{Z}_p$. Then, for a fixed term $\tau \in f$, with probability exceeding $3/4$ the following all occur:

(5.7.i) *τ is not in a prime collision for the choice of p;*

(5.7.ii) *τ is not in a substitution collision for any of the $\boldsymbol{s}_i, i \in [n]$ given by (5.14);*

(5.7.iii) *The matrix $\boldsymbol{S}$ (5.16) is nonsingular.*

We will call a prime p and choice of $\boldsymbol{S} \in \mathbb{Z}_p^{n \times n}$ a **good system** for a term $\tau \in f$ if (5.7.i)-(5.7.iii); otherwise we call it a **bad system** for τ. A good system for τ is precisely a system that for which the resulting substitutions will reveal $\tau \bmod p$, provided that

We will select ℓ distinct primes $p_1, \ldots, p_\ell \in (\lambda..2\lambda]$, and for each prime p_i we construct a system $\boldsymbol{S}_i = [s_{i,j+k+1}]_{j,k=0}^{n-1}$, where $s_{i,1}, \ldots, s_{i,2n} \in \mathbb{Z}_{p_i}$ are chosen independently and uniformly at random. By setting $\ell \geq 8\ln(8T)$, at least half of the ℓ systems will be good for a fixed term of f with probability at least $1 - T/8$. By the union bound every term will have at least $\ell/2$ good systems with probability at least $7/8$.

In order to be able to do this we require that $(\lambda..2\lambda]$ contains at least $8\ln(8T)$ primes. By Proposition 2.4.9, it suffices that $\lambda \geq \frac{64}{5}\ln(8T)\ln(8\ln(8T)) \in \widetilde{\mathcal{O}}(\log(T))$. We require in addition that any $\ell/2$ primes has a product exceeding D, such that we can reconstruct the partial exponents

of f via Chinese remaindering. By Corollary 2.4.5, to achieve this it suffices that $\lambda \geq \frac{10}{3} \ln D$, such that $(\lambda..2\lambda]$ contains at least $2\log_\lambda D$ primes. As $\frac{50}{3}(T-1)\ln D \geq \frac{10}{3}\ln D$ for $T \geq 2$, this is already satisfied by λ given by (5.17).

For $i \in [\ell]$ and $k \in [n]$, we construct the images

$$f_{ik} \stackrel{\text{def}}{=} f(z^{\boldsymbol{s}_{i,k}}) \bmod p \in \mathsf{Z}[z] \bmod p, \quad \text{where} \quad \boldsymbol{s}_{i,k} \stackrel{\text{def}}{=} (s_{i,k}, \ldots, s_{i,k+n-1}) \in \mathbb{Z}_p^n.$$

For each term $\tau \in f$ for which the system $(p_i, \boldsymbol{S}_i)$ is good, we will construct the term $\tau \bmod p$ of

$$f_i \stackrel{\text{def}}{=} f(\boldsymbol{x}) \bmod p_i = f(x_1, \ldots, x_n) \bmod (x_1^{p_i} - 1, \ldots, x_n^{p_i} - 1).$$

As before, need to group terms from the images f_i and f_{ik}, for $i \in [\ell]$ and $k \in [n]$ according to their preimage in f. Each collision polynomial σ, a sum of terms of f, has total degree at most nD. For our purposes, it thus suffices that we choose diversifications from a regular-difference set $\mathcal{U}$ of cardinality $2nD$ or greater. Over $\mathbb{F}_q$, we require that we can work in an extension $\mathbb{F}_{q^r}$, where $r = \lceil \log_q(2nD) \rceil$, such that a random choice of $\boldsymbol{a} \in \mathbb{F}_{q^r}$ will distinguish two distinct preimages of f with probability at least $1/2$. We construct a

As there are in total $(\ell+1)n$ images, we can naively bound the total number of nonzero preimages of terms of f appearing as terms in f_i or f_{ik} by $(\ell+1)nT$. Thus, if we choose $m \geq \lceil \log\left((\ell+1)^2 n^2 T^2 \mu^{-1}\right) \rceil$ diversification vectors $\boldsymbol{a} \in \mathbb{F}_{q^r}$ independently and uniformly at random, then every colliding preimage will be distinguished by one of the choices of $\boldsymbol{a}$ with probability at least $1 - \mu$. That is, the $\boldsymbol{a}$ form a diversifying with high probability. We summarize with the following proposition:

Proposition 5.7.5. *Suppose $f \in \mathbb{F}_q[x_1, \ldots, x_n]$, with $\deg(f) \leq D$ and $\#f \leq T$ terms, and let*

$$\begin{aligned}
\lambda &\geq \lceil \max(21, \quad \tfrac{50}{3}(T-1)\ln D, \quad 10n(T-1), \quad \tfrac{64}{5}\ln(8T)\ln(8\ln(8T))) \rceil, \\
\ell &= \lceil \max(8\ln(8T), \quad 2\log_\lambda D) \rceil, \\
m &= \lceil \log\left((\ell+1)^2 n^2 T^2\right) + 3 \rceil, \\
r &= \lceil \log_q(2nD) \rceil,
\end{aligned}$$

and choose each of the following independently and uniformly at random:

- *ℓ distinct primes $p_1, \ldots, p_\ell \in (\lambda..2\lambda]$;*
- *ℓ Hankel matrices $\boldsymbol{S}_i \in \mathbb{Z}_{p_i}^{n \times n}, i \in [\ell]$;*
- *$\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \in \mathbb{F}_{q^r}^N$.*

Then with probability at least $7/8$, the following all hold:

- *any $\ell/2$ primes p_i have product exceeding D;*

- *every term τ of f, at least half of the systems $(p_i, \mathcal{S}_i), i \in [\ell]$ are good systems for τ;*
- *$\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ forms a diversifying set for f with the images f_i, f_{ik}, $i \in [\ell], j \in [m]$.*

Proposition 5.7.5 helps prove the correctness of Algorithm 19.

Theorem 5.7.6. *Let $f \in \mathbb{F}_q[n]$ be a T-sparse extended-black-box polynomial with partial degress at most D. Then Algorithm 19 interpolates f with probability at least $2/3$ and a query cost of $\widetilde{\mathcal{O}}(n \log T(\log D + \log T))$ $\mathbb{F}_{q^r}$-shifted queries of degree $\mathcal{O}((T+n)\log D)$, where $r = 2nD$.*

Proof. We first prove correctness. The subroutine call to **GetPrimes** succeeds with probability at least $11/12$. By Proposition 5.7.5, we have sufficiently many good systems in order to reconstruct each term $\tau \in f$ with probability at least $7/8$. Thus by the union bound the algorithm succeeds with probability $2/3$ as desired.

We now analyze the query and bit costs. We construct $\ell mn \in \widetilde{\mathcal{O}}(n \log T(\log D + \log T))$ $\mathbb{F}_{q^r}$-shifted queries, each of degree $\mathcal{O}((T+n)\log D)$.

The subroutine call **GetPrimes**$(\lambda, \ell; 1/12)$ costs $\widetilde{\mathcal{O}}(\ell \mathrm{polylog} \lambda)$ bit operations. We solve at most T $n \times n$ Hankel systems modulo $p_i \in \widetilde{\mathcal{O}}(\lambda)$. Accounting for all ℓ primes gives a total bit cost of $\widetilde{\mathcal{O}}(\ell n T \log \lambda)$. Chinese remaindering to construct each of the n components, each at most D, of each of the at-most T exponents of f costs $\widetilde{\mathcal{O}}(nT \log D)$. These costs total to

$$\widetilde{\mathcal{O}}(nT(\ell + \log D)\mathrm{polylog}(\lambda)) \in \widetilde{\mathcal{O}}(nT \log D),$$

as $\ell \in \widetilde{\mathcal{O}}(\log D + \log T)$ and $\lambda \in \widetilde{\mathcal{O}}(nT \log D)$.

The ℓmn images $f_{ikj} \in \mathbb{F}_{q^r}[z]$ may be encoded by their sparse representations each with $\widetilde{\mathcal{O}}(T(\log(q^r) + n \log D)) \in \widetilde{\mathcal{O}}(T(n \log D + \log q))$ bits. Thus all ℓmn images are encoded by

$$\widetilde{\mathcal{O}}(\ell mn T(n \log D + \log q)) \in \widetilde{\mathcal{O}}(n^2 T \log^2 D + n^2 T \log D \log q)$$

bits. Thus the $\widetilde{\mathcal{O}}(n^2 T \log^2 D + n^2 T \log D \log q)$ cost of merely traversing the images f_{ijk} dominates the bit cost of the algorithm. □

5.7.2 A comparison of approaches to interpolate a multivariate SLP

One approach to interpolate an multivariate extended blackbox polynomial $f \in \mathbb{F}_q[\boldsymbol{x}]$ is to combine multivariate and univariate majority-rule algorithms (Algorithms 10 and 18). Per Theorem 5.6.6, multivariate majority-rule interpolation entails some $\mathcal{O}((n + \log T)\log T)$ $\mathbb{F}_{q^r}$-shifted queries of degree $\mathcal{O}(nDT)$, where $r = \lceil \log_q(2nD) \rceil$. If we let $E = nDT$, then by Theorem 3.5.7, each of these queries can be interpolated from $\mathcal{O}(\log E \log T)$ queries of degree $\mathcal{O}(T \log E) \subseteq$

Algorithm 19: Ultravariate majority-rule interpolation

Input: An n-variate extended-black-box polynomial $f \in \mathbb{F}_q[\boldsymbol{x}]$; $D \geq \max_{i=1}^{n} \deg_{x_i}(f)$; $T \geq \#f$.

Output: With probability probability at least $2/3$, a sparse representation for f.

// Choose parameters and make random selections

$\lambda \leftarrow \lceil \max(21, \quad \frac{50}{3}(T-1)\ln D, \quad 10n(T-1), \quad \frac{64}{5}\ln(8T)\ln(8\ln(8T))) \rceil$;

$\ell \leftarrow \lceil \max(8\ln(8T), \quad 2\log_\lambda D) \rceil$;

$m \leftarrow \lceil 2\log(T/2) + 2\log(1+\ell n/2) + 3 \rceil$;

$r \leftarrow \lceil \log_q(2nD) \rceil$;

$p_1, \ldots, p_\ell \leftarrow$ **GetPrimes**$(\lambda, \ell; 1/12)$;

Choose a Hankel matrices $\boldsymbol{S}_i = (\boldsymbol{s}_{i1}^\top, \ldots, \boldsymbol{s}_{in}^\top) \in \mathbb{Z}_{p_i}^{n \times n}$, $i \in [n]$ uniformly at random;

Select $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_{q^r}^n$ independently and uniformly at random;

// Construct images

Interpolate $f_{ik} = f(z^{\boldsymbol{s}_{ik}}) \bmod p_i$ for $i \in [\ell], k \in [n]$;

Interpolate $f_{ikj} = f(\mathbf{a}_j z^{\boldsymbol{s}_{ik}}) \bmod p_i$ for $i \in [\ell], j \in [m], k \in [n]$;

Construct vector polynomials $\boldsymbol{f}_{ik} = (f_{i1k}, \ldots, f_{imk}) \in \mathbb{F}_{q^r}^m[z]$ for $i \in [\ell], k \in [n]$;

// Solve linear systems to construct exponents modulo p_i

for $i \in [\ell]$ **do**

  $\boldsymbol{g}_i \leftarrow 0 \in \mathbb{Z}_{p_i}^m[x_1, \ldots, x_m] \bmod p_i$, a sparse polynomial;

  if $\boldsymbol{S}_i$ *is not invertible modulo* p_i **then continue**;

  for *every* $\boldsymbol{c} \in \mathbb{F}_{q^r}^m$ *such that* $\boldsymbol{c}z^{d_k} \in \boldsymbol{f}_{ik}$ *for some* $d_k \in \operatorname{supp}(\boldsymbol{f}_{ik})$*, and all* $k \in [n]$ **do**

    $\boldsymbol{d} \leftarrow (d_1, \ldots, d_n) \in \mathbb{Z}_{p_i}^n$;

  $\boldsymbol{e}_{\boldsymbol{c},i} \leftarrow$ solution to $n \times n$ Hankel system $\mathcal{S}_i \boldsymbol{e}_{\boldsymbol{c},i} = \boldsymbol{d}_{\boldsymbol{c}}$ over $\mathbb{Z}_{p_i}$;

  $\boldsymbol{g}_i \leftarrow \boldsymbol{g}_i + \boldsymbol{c}x^{\boldsymbol{e}}$;

// Reconstruct exponents via Chinese remaindering

$g \leftarrow 0 \in \mathbb{F}_q[\boldsymbol{x}]$, a sparse polynomial;

for *each* $\boldsymbol{c}$ *appearing as a term* $\boldsymbol{c}\boldsymbol{x}^{\boldsymbol{e}_{\boldsymbol{c},i}} \in \boldsymbol{g}_i$ *for at least* $\ell/2$ $i \in [\ell]$ **do**

  $\boldsymbol{e}_{\boldsymbol{c}} \in [0..D]^n \leftarrow$ solution to congruences $\{\boldsymbol{e}_{\boldsymbol{c}} \equiv \boldsymbol{e}_{\boldsymbol{c},i} (\mathrm{mod} p_i) : \boldsymbol{c}\boldsymbol{x}^{\boldsymbol{e}_{\boldsymbol{c},i}} \in \boldsymbol{g}_i\}$;

  $c \leftarrow$ coefficient of degree $e_{\boldsymbol{c},i,1}$ in f_{i1}, for some $i \in [\ell]$ such that $\boldsymbol{c} \in \operatorname{coeffs}(\boldsymbol{g}_i)$;

  $g \leftarrow g + c\boldsymbol{x}^{\boldsymbol{e}_{\boldsymbol{c}}}$;

return g

$\widetilde{\mathcal{O}}(T \log n \log D)$. This entails $\mathcal{O}((n + \log T) \log T \log D)$ $\mathbb{F}_{q^r}$-shifted queries. Per Lemma 5.1.3, the total $\mathbb{F}_q$-operation cost of these queries on a lenght-L SLP is

$$\widetilde{\mathcal{O}}(L(n + \log T) \log T \log D(T \log n \log D)) \quad \subseteq \quad \widetilde{\mathcal{O}}(LnT \log^2 D),$$

this brings the total bit cost of these queries to

$$\widetilde{\mathcal{O}}(LnT \log^3 D(r \log q)) = \widetilde{\mathcal{O}}(LnT \log^2 D(\log D + \log q)). \tag{5.18}$$

Per Theorem 5.6.6, multivariate majority-rule interpolation costs an additional $\mathcal{O}(n^\omega T \log D)$ bit operations, where $\omega \leq$ is the exponent of matrix multiplication. This brings the total cost to

$$\widetilde{\mathcal{O}}(LnT \log^2 D(\log D + \log q) + n^\omega T \log D).$$

In comparison, the bit cost due to queries of ultravariate majority-rule interpolation is

$$\widetilde{\mathcal{O}}(Ln \log T(\log D + \log T)(T + n) \log D(r \log q)) \\ \subseteq \widetilde{\mathcal{O}}(Ln(n + T) \log^2 D(\log D + \log q)). \tag{5.19}$$

Thus, compared to the first approach, ultravariate majority-rule interpolation adds a soft-oh bit cost factor of $Ln^2 \log^2 D(\log D + \log q)$, and removes a factor $n^\omega T \log D$. In summation, the "ultravariate" approach surpasses the first approach in the case that

$$n \in \widetilde{\Omega}\left(\left(\frac{L \log D(\log D + \log q)}{T}\right)^{1/(\omega-2)}\right).$$

We see that this approach is superior for either n or T sufficiently large.

Chapter 6

Error-correcting sparse interpolation in the Chebyshev basis

6.1 Introduction

In this chapter we present algorithms for the interpolation of a univariate **sparse Chebyshev-basis polynomial**, i.e., a polynomial f that is sparse with respect to the Chebyshev basis. This work appears in [AK15] and is joint work with Erich Kaltofen. We will consider f is given by an *erroneous black box*. That is, we consider f of the form

$$f(x) = \sum_{i=1}^{b} c_i \mathcal{T}_{\delta_i}(x) \in \mathsf{K}[x], \quad 0 \leq \delta_1 > \delta_2 > \cdots > \delta_t, \tag{6.1}$$

where K is a field of characteristic $\neq 2$, $c_j \neq 0$ for all $j \in [t]$, and $\mathcal{T}_n \in \mathsf{K}[x]$ is the nth Chebyshev polynomial of the first kind, defined by:

$$\mathcal{T}_0(x) = 1, \quad \mathcal{T}_1(x) = x, \quad \mathcal{T}_n(x) = 2x\mathcal{T}_{n-1}(x) - \mathcal{T}_{n-2}(x).$$

Recall that an erroneous black-box takes as input $x \in \mathsf{K}$ and produces $f(x) + e(x)$, where $e(x)$ is an error that evaluates to a nonzero value for a bounded number of evaluation points $x \in \mathsf{K}$. We write $a \stackrel{?}{=} f(\omega)$ to denote that a is the result of querying the erroneous black-box polynomial f with ω.

We aim to interpolate f in the presence of errors, while minimizing the number of black-box queries. For the purposes of this chapter we will use bounds $D \geq \deg(f), T \geq t$, and E for the number of errors. Throughout this chapter we let $\mathrm{supp}(f) = \{\delta_1, \ldots, \delta_t\}$.

The erroneous black-box is a useful abstraction for studying interpolation in the presence of unreliable evaluations. Moreover, this problem examines the error-correcting properties of sparse Chebyshev-basis polynomials. By identifying such functions as K-valued vectors indexed by K, we can treat the set of sparse Chebyshev-basis polynomials as an error-correcting code when K is finite, and a generalization thereof when K is infinite.

We give an algorithm that solves this problem for the case $\mathsf{K} = \mathbb{Q}$. To accomplish this, we extend the Prony-like algorithm of Lakshman and Saunders for interpolating a black-box polynomial of the form (6.1). We show that for f comprised of up to three Chebyshev terms, our algorithm requires fewer black-box evaluations than naively applying the Lakshman–Saunders $E + 1$ times on distinct sets of $2T$ evaluation points.

We also give a second algorithm for a black-box sparse Chebyshev polynomial f that reduces the interpolation of a sparse Chebyshev-basis polynomial to sparse black-box interpolation in the monomial basis, and show that this method allows for *early termination*, so that we may probabilistically interpolate f in the case that a bound $T \geq \#f = t$ is not known.

6.2 Background

We give some basic properties of Chebyshev polynomials.

Fact 6.2.1. *Let $m, n \in \mathbb{Z}$. Then the following hold:*

(6.2.1.i) $\begin{bmatrix} 0 & 1 \\ -1 & 2x \end{bmatrix}^n \begin{bmatrix} 1 \\ x \end{bmatrix} = \begin{bmatrix} \mathcal{T}_n(x) \\ \mathcal{T}_{n+1}(x) \end{bmatrix}$.

(6.2.1.ii) $\mathcal{T}_n(\mathcal{T}_m(x)) = \mathcal{T}_{mn}(x) = \mathcal{T}_m(T_n(x))$.

(6.2.1.iii) $\mathcal{T}_m(x)\mathcal{T}_n(x) = \frac{1}{2}(\mathcal{T}_{m+n}(x) + \mathcal{T}_{|m-n|}(x))$.

(6.2.1.iv) $\mathcal{T}_n\left(\frac{1}{2}(x + x^{-1})\right) = \frac{1}{2}\left(x^n + x^{-n}\right)$ *for all* $n \geq 0$, *as an identity in the function field* $\mathsf{K}(x)$.

(6.2.1.v) $\mathcal{T}_n(x) = \frac{1}{2}\left(\left(x - \sqrt{x^2 - 1}\right)^n + \left(x + \sqrt{x^2 - 1}\right)^n\right)$, *as an identity in the quadratic extension* $\mathsf{K}(x, y)/\langle y^2 - x^2 + 1\rangle$ *of the function field* $\mathsf{K}(x)$.

(6.2.1.vi) *For* $\mathsf{K} = \mathbb{R}$ *and* $|\xi| \geq 1$, $\mathcal{T}_m(\xi) \neq 0$.

We cite a Chebyshev analogue of Descartes' rule of signs due to Obrechkoff, that gives an upper bound on the number of real roots ≥ 1 for sparse Chebyshev-basis polynomials over $\mathbb{R}$.

Theorem 6.2.2 (Obrechkoff, 1918). *Let the sequence of polynomials $\{\mathcal{F}_n(x)\}_{n=0}^{\infty}$ be defined by $\mathcal{F}_{-1}(x) = 0$, $\mathcal{F}_0(x) = 1$, and the recurrence relation*

$$x\mathcal{F}_n(x) = \alpha_n\mathcal{F}_{n+1}(x) + \beta_n\mathcal{F}_n(x) + \gamma_n\mathcal{F}_{n-1}(x), n \geq 0,$$

where $\alpha_n, \beta_n, \gamma_n \in \mathbb{R}$, $\alpha, \gamma_n > 0$. Let $(c_1, \ldots, c_t)$ be a list of nonzero real numbers with s sign changes between consecutive values, and $0 < \delta_1 < \delta_2 < \cdots < \delta_t \in \mathbb{R}$. Then $\sum_{i=1}^{t} c_i\mathcal{F}_{\delta_i}(x)$ has at most s roots in (ζ_n, ∞), where ζ_n denotes the largest real root of $\mathcal{F}_n$.

See [DR09] for a proof of Obrechkoff's Theorem. Combined with Fact (6.2.1.vi) this gives the following corollary:

Corollary 6.2.3. *Let $\mathsf{K} = \mathbb{R}$ and $f(x) = \sum_{i=1}^{t} c_i\mathcal{T}_{\delta_i}(x)$, with $\delta_i < \delta_j$ and $c_i \neq 0$ for $1 \leq i < j \leq t$. Then f has at most $t - 1$ distinct real roots ≥ 1.*

As $\mathcal{T}_n(\xi) > \mathcal{T}_m(\xi)$ for $\xi > 1$ and $n > m$, we have in addition the following:

Corollary 6.2.4. *Let $\xi > 1$ and $f(x)$ be a t-sparse polynomial over $\mathbb{R}$ in the Chebyshev basis. Let $m_1 < m_2 < \cdots < m_t \in \mathbb{Z}_{\geq 0}$. If $f(\mathcal{T}_{m_i}(\xi)) = 0$ for all i, then $f(x)$ is identically zero.*

Corollary 6.2.5. *Let $\xi > 1$ and $f(x), g(x)$ be two T-sparse Chebyshev-basis polynomials. Let $m_1 < m_2 < \cdots < m_{2T} \in \mathbb{Z}_{\geq 0}$. If $f(\mathcal{T}_{m_i}(\xi)) = g(\mathcal{T}_{m_i}(\xi))$ for all i, then $f = g$.*

Corollary 6.2.4 gives a polynomial identity test for a black-box sparse Chebyshev-basis polynomial $f \in \mathbb{R}[x]$

Corollary 6.2.6. *Suppose $f \in \mathbb{R}[x]$ is a T-sparse Chebyshev-basis polynomial given by an erroneous black-box that produces at most E errors. Let $\omega_1, \ldots, \omega_{2T+2E} > 1$ be distinct, and, for $i \in [2T + 2E]$, let $a_i \stackrel{?}{=} f(\omega_i)$ be the value produced by the erroneous black-box when queried with ω_i. If g is a T-sparse Chebyshev-basis polynomial such that $g(\omega_i) = a_i$ for all but at most E indices $i \in [2T + 2E]$, then $f = g$.*

6.2.1 Error-correcting sparse interpolation in the monomial basis

In [Shp01] and with Winterhof in [SW05], Shparlinski studied the problem of "noisy" interpolation of sparse polynomials over finite fields, where we are given the support of some B-sparse $f \in \mathbb{Z}_p[x]$, and black-box evaluations of f roughly up to k bits precision, which we will call a **k-bit query**. More precisely, this problem considers an oracle that, given $u \in \mathbb{Z}_p$, produces $v \in \mathbb{Z}$ such that $|f(u) - v| \leq p/2^{k+1}$. They showed that if f is T-sparse with degree at most $p\log(T\log p)^{1-\epsilon}/\log p$, for a constant $\epsilon > 0$, that f can be recovered from $\mathcal{O}((T\log p)^{1+\epsilon}\log p)$ k-bit queries, where $k \in \Theta((T\log p)^{1/2})$, in polynomial time. In [SY11], Saraf and Yekhanin

considered the case of interpolating f over an arbitrary finite field $\mathbb{F}_q$, where $\deg(f) < (1-\delta)q$ and the support of f is known, and when the black-box evaluations of f are erroneous for a proportion α of selected inputs. They show, for fixed α and δ, that a T-sparse polynomial can be interpolated from $\mathcal{O}(k)$ queries with high probability, independent of the choice of q.

Error-correcting sparse interpolation in the monomial basis was considered in [CKP12] and [KP14]. Here we suppose f is a T-sparse polynomial under the monomial basis, and that we are given a sequence $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ of the form $a_i = f(\zeta\omega^i)$, where ω is of multiplicative order at least D

In [CKP12], Comer, Kaltofen, and Pernet give an error-correcting Berlekamp–Massey algorithm, that produces the minimal generator of a linearly generated sequence with errors. Suppose $\boldsymbol{a}' = (a'_0, \ldots, a'_N)$ is a linearly generated sequence with a minimal generator Φ with length at most T, and $\boldsymbol{a} = (a_0, \ldots, a_N)$ is a sequence that disagrees with $\boldsymbol{a}'$ in at most E erroneous locations. The error-correcting Berlekamp–Massey algorithm takes $\boldsymbol{a}$ of length $N \geq 2T(2E+1)$, E and T as inputs, and produces Φ, which they show to be unique. It does this by way of a simple majority-rule procedure that runs Berlekamp–Massey on $2E+1$ contiguous subsequences of $\boldsymbol{a}$ of length $2T$, taking Φ to be the minimal generator corresponding produced by the majority of the "blocks". Using this approach, they are able to identity the erroneous components of $\boldsymbol{a}$ and produce the corrected sequence $\boldsymbol{a}'$. If $\boldsymbol{a}'$ is a sequence of evaluations of the form $a'_i = f(\zeta\omega^i)$, where ω has multiplicative order exceeding D and f is T-sparse under the monomial basis, then f may then be obtained from $\boldsymbol{a}'$ via Prony's algorithm. We call this method **block-decoding Prony**.

They show, moreover, that if $N < 2T(2E+1)$, then one can construct examples whereby Φ is not unique.

Example 6.2.7 (Example 2.1, [CKP12])**.** *Suppose* K *is a field with* $\mathrm{char}(\mathsf{K}) \neq 2$*, and consider the sequence* $(a_0, \ldots, a_{2T(2E+1)-1})$ *given by*

$$a_i = \begin{cases} 1 & \text{if } i \bmod 8T \in \{2T-1, 4T-1, 6T-1\}, \\ -1 & \text{if } i \bmod 8T = 8T-1, \\ 0 & \text{otherwise}, \end{cases}$$

then $\boldsymbol{a}$ *differs in exactly* E *locations from two distinct linearly generated sequences* $\boldsymbol{a}^+$ *and* $\boldsymbol{a}^-$ *with minimal generators* $y^T + 1$ *and* $y^T - 1$ *respectively, and given by*

$$a_i^{\pm} = \begin{cases} (-1)^{\mp 1} & \text{if } i \bmod 8T = 8T-1, \\ a_i & \text{otherwise}. \end{cases}$$

If N instead is only at least $2T(E+1)$, then we can produce a list containing $E+1$ candidates for the appropriate minimal generator Φ, by running Berlekamp–Massey on $E+1$ contiguous blocks of length $2T$. By running Prony's method on each of these blocks, we obtain a list of at

most $E+1$ possibilities for f, that is guaranteed to contain the actual interpolant f. We will call this method **list-block-decoding Prony**. In the case that $\mathsf{K} = \mathbb{R}$, we can determine the unique interpolant via Descartes' rule of signs. That is, if we choose all the evaluation points $\zeta\omega^i$ to be positive, then any s-sparse polynomial will have at most $s-1$ positive roots, such that any T-sparse g that is not the true T-sparse interpolant f will agree with f in less than $2T$ places. Thus, provided $N \geq 2E+2T$, the true interpolant f will be the unique T-sparse polynomial for which $f(\zeta\omega^i) \neq a_i$ in at most E locations $i \in [0..N)$.

In [KP14], Kaltofen and Pernet give an improved list-decoding interpolation algorithm. They run Prony's algorithm on every length-$2B$ subsequence of $\boldsymbol{a}$ that is the evaluation of f over a nontrivial geometric sequence. These correspond to all length-$2T$ **affine subsequences** of $\boldsymbol{a}$, which we define as subsequences indexed by a nontrivial arithmetic progression, i.e., $(a_{u+iv})_{i=0}^{k-1}$, where $0 \leq u < u+(2T-1)v < N$. We call this method **list-decoding Prony**. They completely characterize the instances for which list-decoding Prony outperforms block-list-decoding Prony in terms of the number of black-box queries required. Namely, they show the following:

Lemma 6.2.8. *Let $\boldsymbol{a} = (a_0, a_1, \dots)$ denote a sequence containing at most E errors, and let $N_{k,E}$ denote the minimum value of N for which $(a_0, \dots, a_{N-1})$ must contain a length-k affine subsequence that does not contain an error, then we have that $N_{k,E} \leq k(E+1)$. Moreover, $N_{k,E} = k(E+1)$ if and only if $E+2 \leq p_k$, where p_k is the least prime factor of k.*

They also compute $N_{k,E}$ for $k \leq 13$ and $E \leq 10$, as well as other select values. The value $N_{k,E}$ is related to the problem of finding the largest subsequence of $\{1, 2, \dots, N\}$ not containing k terms in an arithmetic progression. We define $r(k,N)$ to be the cardinality of the largest such subsequence. We have that

$$N_{k,E} = \min\{N : N - r(k,N) \geq E+1\}.$$

Erdős and Turán studied $r(k,E)$ in the case that $k=3$ [ET36]. They conjectured the following result that was later proven by Szemereédi.

Theorem 6.2.9 ([Sze75])**.** *$r(k,N) \in o(N)$ for fixed $k \in \mathbb{Z}_{>0}$.*

This alone is insufficient to derive stronger asymptotic bounds on $N_{k,E}$, which remains an open problem.

6.2.2 The Lakshman–Saunders algorithm

In [LS95], Lakshman and Saunders give a Prony-like algorithm for a sparse Chebyshev-basis polynomial $f \in \mathbb{R}[x]$ of the form (6.1). Their algorithm evaluated f over points of the form $T_i(\xi)$,

where $\xi > 1$. Their algorithm was designed when t was known; however, it is easy to adapt to the case that we are given a bound $T \geq t$.

They show that the evaluations $a_i = f(T_i(\xi)), i \in [0..2T]$ forms a Hankel-plus-Toeplitz system whose solution encodes the Chebyshev-basis support of f, similar to Prony's algorithm. The solution $\boldsymbol{\phi}$ to the Hankel-plus-Toeplitz system

$$\underbrace{\begin{bmatrix} 2a_0 & 2a_1 & \cdots & 2a_{t-1} \\ 2a_1 & a_2 + a_0 & \cdots & a_t + a_{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ 2a_{t-1} & a_t + a_{t-2} & \cdots & a_{2t-2} + a_0 \end{bmatrix}}_{\stackrel{\text{def}}{=} \boldsymbol{A}^{(t)}} \underbrace{\begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{t-1} \end{bmatrix}}_{\stackrel{\text{def}}{=} \boldsymbol{\phi}} = \underbrace{\begin{bmatrix} 2a_t \\ a_{t+1} + a_{t-1} \\ \vdots \\ a_{2t-1} + a_1 \end{bmatrix}}_{\stackrel{\text{def}}{=} \boldsymbol{\alpha}}, \tag{6.2}$$

forms the coefficients of the auxillary polynomial $\Phi \in \mathbb{R}[y]$ whose roots give the Chebyshev-basis support of f,

$$\Phi = \sum_{i=0}^{t} \phi_i y^i = \prod_{i=1}^{t} (y - T_{\delta_j}(\xi)), \tag{6.3}$$

where we define $\phi_t = 1$. Moreover, if we define

$$\mathbf{A}_{(s)} \stackrel{\text{def}}{=} \left[a_{i+j} + a_{|i-j|} \right]_{i,j=0}^{s-1}, \quad \text{for } s \in [1..T], \tag{6.4}$$

then it is easy to show that $\mathbf{A}^{(s)}$ is singular for $s > t$ (Lemma 6.3.2. We thus can obtain the sparsity t of f as the greatest integer $s \leq T$ for which $\mathbf{A}^{(s)}$ is nonsingular, or 0 if no such integer s exists.

As $T_\delta(\xi)$ is strictly increasing with respect to δ for $\xi > 1$, we can find δ from T_δ by way of a binary search on $[0..D]$. $T_\delta(\xi)$ can be computed for various powers of δ efficiently by a square-and-multiply approach, using Fact (6.2.1.i).

Once we have the support $\delta_1, \ldots, \delta_t$, we can obtain the coefficients c_i by way of the linear system

$$\begin{bmatrix} T_{\delta_1}(T_0(\xi)) & T_{\delta_2}(T_0(\xi)) & \cdots & T_{\delta_t}(T_0(\xi)) \\ T_{\delta_1}(T_1(\xi)) & T_{\delta_2}(T_1(\xi)) & \cdots & T_{\delta_t}(T_1(\xi)) \\ \vdots & \vdots & \ddots & \vdots \\ T_{\delta_1}(T_{t-1}(\xi)) & T_{\delta_2}(T_{t-1}(\xi)) & \cdots & T_{\delta_t}(T_{t-1}(\xi)) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{bmatrix} \tag{6.5}$$

Algorithm 20 describes this approach in pseudocode.

Algorithm 20: The Lakshman–Saunders algorithm for the interpolation of a sparse Chebyshev-basis black-box rational polynomial

Input: A black-box polynomial $f \in \mathbb{Q}[x]$; $T \geq \#f$; $\xi \in \mathbb{Q}, \xi > 1$.
Output: A sparse Chebyshev-basis representation of f

1 **for** $i \leftarrow 0$ **to** $2T$ **do** $a_i \leftarrow f(T_i(\xi))$;
2 $t \leftarrow \max\{s \in [0..T] : \boldsymbol{A}^{(s)}$ defined by (6.4) is nonsingular$\}$;
3 Obtain $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{t-1})$ as solution to Hankel-plus-Toeplitz system (6.2);
4 Factor $\Phi = y^t + \sum_{i=0}^{t-1} \phi_i y^i$ to obtain the roots $\mathcal{T}_{\delta_i}(\xi), i \in [t]$;
5 Find the indices δ_i from $\mathcal{T}_{\delta_i}(\xi)$, for $i \in [t]$;
6 Compute coefficients $\boldsymbol{c} = (c_1, \ldots, c_t)$ as solution to linear system (6.5);
7 **return** $\sum_{i=1}^{t} c_i \mathcal{T}_{\delta_i}$;

6.3 List-decoding Lakshman–Saunders algorithm

In this section we descibe the **list-decoding Lakshman–Saunders** algorithm, that is an analogue to list-decoding Prony's method. Whereas list-decoding Prony considered subsequences of the form $a_{u+iv}, i \in [0..2T)$, we will consider subsequences of the form

$$\begin{gathered}(a_{|u-Tv|}, a_{|u-(T-1)v}, \ldots, a_{|u+(2T-1)v|}), \text{ where} \\ a_i = f\big(\mathcal{T}_i(\xi)\big) \text{ for some } \xi \in \mathbb{Q}, \xi > 1, \text{ and} \\ |u+iv| \neq |u+jv| \text{ for } i \neq j \in [0..T).\end{gathered} \tag{6.6}$$

We call such subsequences **valid folded affine subsequences**. These subsequences will comprise at least $2T$ and as many as $3T$ entries of $\boldsymbol{a}$. The distinctness constraint in (6.6) is equivalent to $u \neq -kv/2$ for $k \in [1..2T-2)$.

Throughout Section 6.3, we will suppose $f \in \mathbb{Q}[x]$ is of the form (6.1), and we consider a sequence of evaluations $\boldsymbol{a}$ given by (6.6) with $\xi \in \mathbb{Q}$, $\xi \geq 1$.

6.3.1 Generalizing Laskhman–Saunders to folded affine subsequences

For the purposes of Section 6.3.1, fix $u, v \in \mathbb{Z}$ and $\xi \in \mathbb{Q}$ with $v, \xi \geq 1$. We will also define $\Phi \in \mathbb{Q}[y]$ here by

$$\Phi = \prod_{j=1}^{t} \big(y - \mathcal{T}_{v\delta_j}(\xi)\big) = \sum_{j=1}^{t} \phi_j y^j.$$

We first prove a generalization of Lemma 5 of [LS95], whose proof is similar.

Lemma 6.3.1. *Fix $v \in \mathbb{Z}_{>0}, \xi \in \mathbb{Q}_{\geq 1}$, and let $\Phi \in \mathbb{Q}[y]$ be given by (6.3), with $\phi_t = 1$. Then, for $u, v \in \mathbb{Z}$,*

$$\sum_{j=0}^{t} \phi_j (a_{|u+(i+j)v|} + a_{|u+(i-j)v)} = 0. \tag{6.7}$$

Proof. Using Facts (6.2.1.ii) and (6.2.1.iii), we have that

$$\begin{aligned} \sum_{j=0}^{t} \phi_j a_{u+(i+j)v} &= \sum_{j=0}^{t} \phi_j \left(\sum_{\ell=1}^{t} c_\ell \mathcal{T}_{\delta_\ell}\big(\mathcal{T}_{u+(i+j)v}(\xi)\big) \right), \\ &= \sum_{\ell=1}^{t} c_\ell \left(\sum_{j=0}^{t} \phi_j \mathcal{T}_{u+(i+j)v}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) \right), \end{aligned} \tag{6.8}$$

and

$$\begin{aligned} \mathcal{T}_{u+(j+i)v}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) &= 2\mathcal{T}_{vj}\big(\mathcal{T}_{\delta_\ell}(\xi)\big)\mathcal{T}_{u+iv}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) - \mathcal{T}_{|u+(i-j)v|}\big(\mathcal{T}_{\delta_\ell}(\xi)\big), \\ &= 2\mathcal{T}_{j}\big(\mathcal{T}_{v\delta_\ell}(\xi)\big)\mathcal{T}_{u+iv}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) - \mathcal{T}_{|u+(i-j)v|}\big(\mathcal{T}_{\delta_\ell}(\xi)\big). \end{aligned} \tag{6.9}$$

Thus we can rewrite the inner sum of (6.8) as

$$\sum_{j=0}^{t} \phi_j \Big(2\mathcal{T}_j\big(\mathcal{T}_{v\delta_\ell}(\xi)\big)\mathcal{T}_{u+iv}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) - \mathcal{T}_{|u+(i-j)v|}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) \Big),$$

$$= 2\underbrace{\Phi\big(\mathcal{T}_{v\delta_\ell}\big)}_{=0}\mathcal{T}_{ui+v}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) - \sum_{j=0}^{t} \phi_j \mathcal{T}_{|u+(i-j)v|}\big(\mathcal{T}_{\delta_\ell}(\xi)\big).$$

Thus (6.8) becomes

$$\begin{array}{rcl} -\sum_{\ell=1}^{t} c_\ell \left(\sum_{j=0}^{t} \phi_j \mathcal{T}_{|u+(i-j)v|}\big(\mathcal{T}_{\delta_\ell}(\xi)\big) \right) & = & -\sum_{j=0}^{t} \phi_j \sum_{\ell=1}^{t} c_\ell T_{\delta_\ell}\big(T_{|u+(i-j)v|}(\xi)\big), \\ = \; -\sum_{j=0}^{t} \phi_j f\big(T_{|u+(i-j)v|}(\xi)\big) & = & -\sum_{j=0}^{t} \phi_j a_{|u+(i-j)v|}. \end{array}$$

The identity (6.7) follows. □

Taking (6.7) for all $i \in [0..t)$ gives a Hankel-plus-Toeplitz system $\boldsymbol{A}^{(t,u,v)}\boldsymbol{\phi} = \boldsymbol{\alpha}$, where

$$\begin{aligned} \boldsymbol{\phi} &= (\phi_0, \ldots, \phi_{t-1}) \\ \boldsymbol{A}^{(s,u,v)} &= \big[a_{|u+(i+j)v|}\big]_{i,j=0}^{s-1} + \big[a_{|u+(i-j)v}\big]_{i,j=0}^{s-1} \qquad \text{for } s \in [1..T], \\ \boldsymbol{\alpha} &= \big[a_{|u+(i+t)v|} + a_{|u+(i-t)v|}\big]_{i=0}^{t-1}. \end{aligned} \tag{6.10}$$

Thus we can obtain $\boldsymbol{\phi}$ from $\boldsymbol{A}$ and $\boldsymbol{\alpha}$, provided $\boldsymbol{A}$ is nonsingular. To that end we adapt Lemma 6 of [LS95] to our generalized setting.

Lemma 6.3.2. *Let $u, v \in \mathbb{Z}$ with $v > 0$. If the values $|u + iv|$, for $i \in [0..t)$ are distinct, then $\boldsymbol{A}^{(t,u,v)}$ is nonsingular. Moreover, $\boldsymbol{A}^{(s,u,v)}$ is singular for $s > t$.*

Proof. $\boldsymbol{A}^{(s,u,v)}$ admits a factorization $\boldsymbol{A}^{(s,u,v)} = \boldsymbol{M}^{(s)}\boldsymbol{D}\boldsymbol{N}^{(s)}$, where $\boldsymbol{D} = \operatorname{diag}(2c_1, \ldots, 2c_t)$ and $\boldsymbol{M}^{(s)} \in \mathbb{Q}^{s\times t}$, $\boldsymbol{N}^{(s)} \in \mathbb{Q}^{t\times s}$ are given by

$$M_{ij}^{(s)} = T_{|u+iv|}\big(T_{\delta_{j+1}}(\xi)\big) \qquad\qquad N_{ij}^{(s)} = T_{|jv|}\big(T_{\delta_{i+1}}(\xi)\big). \tag{6.11}$$

To show this, using Facts (6.2.1.ii) and (6.2.1.iii), we have that the (i, j)th entry of $\boldsymbol{M}\boldsymbol{D}\boldsymbol{N}$ is

$$\begin{aligned}
&\sum_{\ell=1}^{t} 2c_\ell T_{|u+iv|}\big(T_{\delta_\ell}(\xi)\big) \cdot T_{jv}\big(T_{\delta_\ell}(\xi)\big),\\
=&\sum_{\ell=1}^{t} c_\ell \left(T_{|u+(i+j)v|}\big(T_{\delta_\ell}(\xi)\big) + T_{|u+(i-j)v|}\big(T_{\delta_\ell}(\xi)\big)\right),\\
=&\sum_{\ell=1}^{t} c_\ell \left(T_{\delta_\ell}\big(T_{|u+(i+j)v|}(\xi)\big) + T_{\delta_\ell}\big(T_{|u+(i-j)v|}(\xi)\big)\right)\\
=&f\big(T_{|u+(i+j)v|}(\xi)\big) + f\big(T_{|u+(i-j)v|}(\xi)\big)\\
=&a_{|u+(i+j)v|} + a_{|u+(i-j)v|}.
\end{aligned}$$

This proves singularity for the case $s > t$. Now suppose $s = t$ and let $\boldsymbol{b} = (b_0, \ldots, b_{t-1})$ be such that $\boldsymbol{b}^\top \boldsymbol{M}^{(t)} = \boldsymbol{0}$. Then $\sum_{i=0}^{t-1} b_i T_{|u+iv|}$ is a b-sparse Chebyshev-basis polynomial with roots $T_{\delta_1}(\xi), \ldots, T_{\delta_b}(\xi)$. By Corollary 6.2.4, $\boldsymbol{b}$ is necessarily zero. It follows that $\boldsymbol{M}$ is nonsingular. $\boldsymbol{N}^{(t)}$ is nonsingular by a similar argument. This completes the proof. □

The coefficients can then be obtained from the system

$$\begin{bmatrix}
T_{\delta_1}\big(T_{|u|}(\xi)\big) & \cdots & T_{\delta_t}\big(T_{|u|}\big)\\
T_{\delta_1}\big(T_{|u+v|}(\xi)\big) & \cdots & T_{\delta_t}\big(T_{|u+v|}\big)\\
\vdots & \ddots & \vdots\\
T_{\delta_1}\big(T_{|u+(t-1)v|}(\xi)\big) & \cdots & T_{\delta_t}\big(T_{|u+(t-1)v|}\big)
\end{bmatrix}
\begin{bmatrix} c_1\\ c_2\\ \vdots\\ c_t \end{bmatrix} =
\begin{bmatrix} a_{|u|}\\ a_{|u+v|}\\ \vdots\\ a_{|u+(t-1)v|} \end{bmatrix}. \tag{6.12}$$

The matrix appearing above in (6.12) is exactly $\boldsymbol{M}^{(t)}$ by Fact (6.2.1.ii), and hence is nonsingular.

Thus we can essentially run Lakshman–Saunders over any sequence of the form (6.6). This is described in Algorithm 21.

Algorithm 21: The Lakshman–Saunders algorithm for folded affine subsequences

Input: A black-box polynomial $f \in \mathbb{Q}[x]$; $T \geq \#f$; $\xi \in \mathbb{Q}, \xi \geq 1$; $u, v \in \mathbb{Z}$ where $v > 0$ and $|u + iv| \neq |u + jv|$ for $i \neq j \in [0..t] =$).

Output: A sparse Chebyshev-basis representation of f

1 **for** $i \leftarrow -T$ **to** $2T - 1$ **do** $a_{|u+iv|} \leftarrow f(T_{|u+iv|}(\xi))$;
2 $t \leftarrow \max\{s \in [0..T] : \boldsymbol{A}^{(s,u,v)}$ defined by (6.4) is nonsingular$\}$;
3 Obtain $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{t-1})$ as solution to Hankel-plus-Toeplitz system $\boldsymbol{A}\boldsymbol{\phi} = \boldsymbol{\alpha}$ with $\boldsymbol{A}(t, u, v), \boldsymbol{\alpha}^{(t)}$ given by (6.10);
4 Factor $\Phi = y^t + \sum_{i=0}^{t-1} \phi_i y^i$ to obtain the roots $\mathcal{T}_{v\delta_i}(\xi), i \in [t]$;
5 Find the indices δ_i from $\mathcal{T}_{v\delta_i}(\xi)$, for $i \in [t]$;
6 Compute coefficients $\boldsymbol{c} = (c_1, \ldots, c_t)$ as solution to linear system (6.5);
7 **return** $\sum_{i=1}^{t} c_i \mathcal{T}_{\delta_i}$;

6.3.2 A list-decoding algorithm

In this section we present our simple list-decoding Lakshman–Saunders algorithm. We merely produce a sequence of N erroneous black-box evaluations $a_i \stackrel{?}{=} f(\mathcal{T}_i(\xi))$ for $i \in [0..N)$, and run Algorithm 21 over all length-$3T$ valid folded affine subsequences $(a_{|u+iv|})_{i=-T}^{2T-1}$, with $|u + iv| \neq |u + jv|$ for $i \neq j \in [0..T)$. Without loss of generality, we can take $|u| < |u + (T - 1)v|$, or equivalently, $u \geq (T - 1)v/2$, as $\{u + iv : i \in [-T..2T)\}$ and $\{u' + iv : i \in [-T..2T)\}$ agree for $u' = -(u + (T - 1)v)$. For each such pair (u, v), Algorithm 21 produces a candidate sparse representation for the interpolant f, which it tests to see if it is the unique interpolant.

Algorithm 22: List-decoding Lakshman–Saunders

Input: An erroneous black-box polynomial $f \in \mathbb{Q}[x]$; $T \geq \#f$; $N \geq 2T + 2E$.

Output: If $[0..N)$ contains a subset length-$3T$ valid folded arithmetic progression P for which each $a_i, i \in P$, avoids errors, then Algorithm 22 produces f. Otherwise it produces `fail`.

1 **for** $v \leftarrow 1$ **to** $\lfloor \frac{2N-1}{3T-1} \rfloor$ **do**
2 **for** $u \leftarrow \lceil \frac{1}{2}(T - 1) \rceil$ **to** $N - 1 - (3T - 1)v$ **do**
3 **if** $|u + iv| = |u + jv|$ *for some* $i \neq j \in [0..T)$ **then continue**;
4 $f \leftarrow$ output of Algorithm 21 on inputs f, T, ξ, u, v;
5 **if** $f(T_i(\xi)) = a_i$ *for all but at most* E *exceptions in* $[0..N)$ **then**
6 **return** f;
7 **return** `fail`;

If N is sufficiently large that any set of E error locations in $[0..N)$ will avoid at least one valid length-$3T$ arithmetic progression, then Algorithm 22 will produce a unique interpolant. If, in addition, $N \geq 2T + 2E$, then we can identify this interpolant per Corollary 6.2.6.

6.3.3 Upper-bounding the query cost to interpolate T-sparse f for $T \leq 3$

We would like that $N^*_{3T,E} \leq 2T(E+1)$, or can instead interpolate f with fewer erroneous black-box evaluations by running Algorithm 20 $E+1$ times, using in each iteration a set of $2T$ evaluation points distinct from those used in previous iterations. We call this approach **block-list-decoding Lakshman–Saunders**, which is an analogue of block-list-decoding Prony.

An open problem remains whether, for each $T \in \mathbb{Z}$, there exists E_T such that we can interpolate f in fewer than $2T(E+1)$ queries for all $E \geq E_T$. We determined this to be true in the cases that $T = 1, 2, 3$, via the following strategy.

Consider a block of evaluations of the form $(a_1, \ldots, a_M)$, i.e., excluding the evaluation $a_0 = f(1)$. We can interpolate f from such a block of evaluations provided it contains affine subsequence of length $3T$ that does not intersect with any errors.

For a fixed T, we determine a pair (M_T, E_T), such that for $M = M_T$, such a block of evaluations with at most E_T errors necessarily contains such a subsequence. Then, for an arbitrary error bound $E > E_T$, we run Algorithm 21 on blocks of $\lceil E/E_T \rceil$ non-intersecting evaluation points.

If we do this for $\lceil (E+1)/(E_T+1) \rceil$ blocks, then one of the blocks must contain at most E_T errors and will give us f, or else each block would have at least $E_T + 1$ errors, such that the total number of errors would be at least $E + 1$. The total number of evaluations by this approach is at most $M_T \lceil (E+1)/(E_T+1) \rceil$.

From Table 1 of [KP14], we have $N_{3,8} = 17$, such that we can interpolate 1-sparse f from a block of $M_1 = 17$ evaluations with at most $E_1 = 8$ errors. Thus, for arbitrary E, we can interpolate f from $17\lceil (E+1)/9 \rceil$ evaluations. A straightforward computation shows this is less than $2T(E+1)$ for $T = 1$ and $E \geq 153$.

Similarly, we have $N_{6,8} = 34$, such that we can interpolate 2-sparse f from a block of $M_2 = 34$ evaluations with at most $E_2 = 8$ errors. This gives a method with $34\lceil (E+1)/9 \rceil$ evaluations, which is less than $2T(E+1)$ for $T = 2$ and $E \geq 136$.

Clément Pernet has communicated to us that $N_{9,12} = 74$. Thus we can interpolate 3-sparse f in the presence of E errors with $74\lceil (E+1)/13 \rceil$ evaluations, which is less than $2T(E+1)$ for $T = 3$ and $E \geq 403$.

6.4 Reducing sparse-Chebyshev interpolation to sparse-monomial interpolation

In this section we present an algorithm for the sparse interpolation of a black-box polynomial $f \in \mathsf{K}[x]$ given by (6.1) when K is a field of characteristic not 2, and $c_j \neq 0$ for all $1 \leq j \leq t$, from evaluations of the from

$$a_i = f(\tfrac{\omega^i+\omega^{-i}}{2}), \text{where } \omega \in \mathsf{K}, \omega \neq 0, \tag{6.13}$$

where we assume throughout 6.4 that the values

$$\omega^{-2\delta_1}, \ldots, \omega^{-2\delta_t}, \omega^{2\delta_t}, \ldots, \omega^{2\delta_1}$$

are all distinct, with the possible exception that $\omega^{2\delta_t} = \omega^{-2\delta_t}$ in the case that $\delta_t = 0$. In this section we stress that f is not an erroneous black-box polynomial, i.e., all the evaluations are correct. Note that $a_i = f(T_i(\xi))$ with $\xi = (\omega + \omega^{-1})/2$, so the evaluations used by Lakshman–Saunders and its variants apply.

We will show how to interpolate f from a worst-case $2T$ evaluations. Our method reduces the interpolation of a T-sparse Chebyshev-basis polynomial to that of a $2T$-sparse monomial-basis Laurent polynomial. In Section 2.8, we adapt the algorithm to early termination, such that in a sufficiently large field K and with high probability, we only require $2t + 1$ evaluations.

Define the Laurent polynomial

$$g(z) \stackrel{\text{def}}{=} f(\tfrac{z+z^{-1}}{2}) = \sum_{j=1}^{t} \tfrac{c_j}{2}(z^{\delta_j} + z^{-\delta_j}) \in \mathsf{K}[z, z^{-1}].$$

g is $2t$-sparse if $\delta_t > 0$ and $(2t-1)$-sparse otherwise. Observe that g is **reciprocal**, i.e., its coefficients read the same way in orders of increasing or decreasing degree.

$$a_i = a_{-i} = g(\omega^i), \quad \text{for } i \in \mathbb{Z}.$$

We can interpolate g via Prony's algorithm. As g is $2T$-sparse, we can interpolate g from the $4T$ evaluations

$$(b_0, \ldots, b_{4T-1}) = (a_{-2T-1}, a_{-2T-3}, \ldots, a_{-1}, a_1, \ldots, a_{2T-1}),$$

which can be produced from $2T$ evaluations of f. We do not use the evaluation a_0 as it does not give us a free additional evaluation of g.

By the theory of Prony's method, we know the minimal generator of the sequence $\boldsymbol{b}$, which we cite as a lemma.

Lemma 6.4.1. *Suppose ω has multiplicative order exceeding $2D+1$. Then the minimal generator of the sequence $\boldsymbol{b} \in \mathsf{K}^{\mathbb{Z}}$ (6.13) is*

$$\Phi = \begin{cases} \prod_{j=1}^{t}(y-\omega^{2\delta_j})(y-\omega^{-2\delta_j}) & \text{if } \delta_t > 0 \\ (y-1)\prod_{k=2}^{t}(y-\omega^{2\delta_j})(y-\omega^{-2\delta_j}) & \text{if } \delta_t = 0 \end{cases}$$

Thus the minimal generator Φ encodes the support of g and thus f as well. The corresponding coefficients of f may be obtained via the $t \times t$ linear system (6.12) with $u = 1$ and $v = 2$, which reuses the evaluations $a_1, a_3, \ldots, a_{2t-1}$.

As with Prony's algorithm, we can employ the technique of early termination in this setting as well. This is useful in the case that a bound $T \geq t$ is not provided, such that we can probabilistically estimate t, or if we want to interpolate f with the computation and query costs dependent on the true sparsity t and not a bound T.

Namely, if we define the Hankel matrices

$$\boldsymbol{H}_s = \begin{bmatrix} a_{-2s-1} & a_{-2s+1} & \cdots & a_{-1} \\ a_{-2s+1} & a_{-2s+3} & \cdots & a_{1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{-1} & a_{1} & \cdots & a_{2s-3} \end{bmatrix}, \qquad s \in \mathbb{Z}_{\geq 0},$$

then we can probabilistically determine t as the least integer s such that $\boldsymbol{H}^{(s+1)}$ is singular. If g has absolute degree at most D, then for a random choice of $\omega \in \mathsf{K}$, we have that from Section 2.7 this approach will give t with probability at least $Dt_g(t_g-1)(t_g+1)/(3\#\mathsf{K})$.

Chapter 7

Fast Fourier-sparsity testing of Boolean functions

In this chapter we consider the problem of **Fourier-sparsity testing**. Specifically, we will consider f acting over the n-dimensional **hypercube**, i.e., $\mathbb{F}_2^n$, where high-order roots of unity are not available. This is based on ongoing work with Eric Blais. One important class of such functions are **Boolean functions**, i.e., functions that take as input a string of n bits and whose output is true or false.

We have seen probabilistic sparsity testing used to expedite Prony's algorithm (via early termination), sparse interpolation of extended black boxes, and sparse multiplication. In each of these cases we use a probabilistic tester with one-sided error, such that we do not erroneously overestimate the sparsity. In each of these cases we restricted our attention to univariate polynomials over domains containing roots of unity of large order.

These algorithms are based on the construction and inspection of low-dimensional images of f. We saw similar techniques used with the technique of randomized Kronecker substitutions in Chapter 5. Such techniques also appear in **Sparse Fourier Transforms (SFTs)**, a family of algorithms that compute the discrete Fourier transform $\widehat{f}$ of f, when it is known to be sparse. More generally, these algorithms may find a t-term approximation that, in terms of ℓ_2 distinct from $\widehat{f}$, is within a constant factor of the *best* t-term approximation $\widehat{f}$. For completeness, we analyze a sparse Fourier transform algorithm based on ideas originating from [GL89] and [Lev93].

7.1 Background

We consider complex-valued functions that act on the **hypercube** $\mathbb{F}_2^n$. The hypercube admits a graph representation whose vertices are the elements of $\mathbb{F}_2^n$, with edges connecting any two

elements of Hamming distance 1 (see Figure 7.1). We identify the additive group structure of $\mathbb{F}_2$ with the multiplicative structure of $\{\pm 1\} \subset \mathbb{R}$, and use both representations throughout. A function $f : \mathbb{F}_2^n \to \mathbb{C}$ admits a **Fourier representation**

$$f = \sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)\chi(\alpha),$$

where the $\widehat{f}(\alpha)$ are the **Fourier coefficients** of f, also known as **Hadamard** or **Walsh–Hadamard coefficients**, and $\chi_\alpha(x) : \mathbb{F}_2^n \to \mathbb{C}$ is the **character** or **parity function** defined by

$$\chi_\alpha(x) = (-1)^{\alpha^\top x} = (-1)^{x^\top \alpha} = \begin{cases} 1 & \text{if } \sum_{i=1}^n \alpha_i x_i = 0 \in \mathbb{F}_2^n, \\ -1 & \text{otherwise.} \end{cases}$$

In other words, $\chi_\alpha(x)$ outputs 1 or -1 according to the parity of x restricted to the support of α. We will generally use w, x, y, z for arguments to f and α, β, κ letters for arguments to $\widehat{f}$. We define the **support** of f to be $\operatorname{supp}(f) \stackrel{\text{def}}{=} \{\alpha \in \mathbb{F}_2^n : \widehat{f}(\alpha) \neq 0\}$.

The character functions form an orthonormal basis for $\mathbb{C}^{\mathbb{F}_2^n}$. We can express the Fourier coefficient $\hat{f}(\alpha)$ as an expectation

$$\widehat{f}(\alpha) = \mathbb{E}_x[f(x)\chi_\alpha(x)] = 2^{-n} \sum_{x \in \mathbb{F}_2^n} f(x)\chi_\alpha(x) \tag{7.1}$$

where $\mathbb{E}_x$ is the expectation with x taken from the uniform distribution over $\mathbb{F}_2^n$. We call a term $\widehat{f}(\alpha)\chi_\alpha$ a **Fourier term** or merely a term. We say the **Fourier mass** of a sum of terms $g = \sum_\alpha \widehat{g}(\alpha)\chi_\alpha$ of Fourier terms is the sum of the square of its Fourier coefficients, i.e., $\sum_\alpha \widehat{g}(\alpha)^2 = \|g\|_2^2$.

The function f admits an equivalent representation

$$f : \{\pm 1\}^n \to \mathbb{C}, \quad x \mapsto \sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha) x_1^{\alpha_1} \cdots x_n^{\alpha_n},$$

so that we may think of the Fourier representation of f as a complex-valued multilinear polynomial over $\mathbb{C}$, but whose evaluations are restricted to the **real-valued hypercube** $\{\pm 1\}^n$. We let $\mathcal{H}$ denote the class of complex-valued functions acting on the n-dimensional hypercube. Since the ℓ^2-norm differs for $f : \mathbb{F}_2^n \to \mathbb{C}$ and $\widehat{f} : \mathbb{F}_2^n \to \mathbb{C}$ by a constant factor, we let $\widehat{\mathcal{H}}$ denote the class $\mathcal{H}$ in the Fourier domain.

One important subclass of functions are **Boolean functions**. Named after the logician George Boole, a Boolean function takes as input a string of n bits and outputs a single bit. We may represent a Boolean function f as a function that acts on the hypercube and produces 1 or -1.

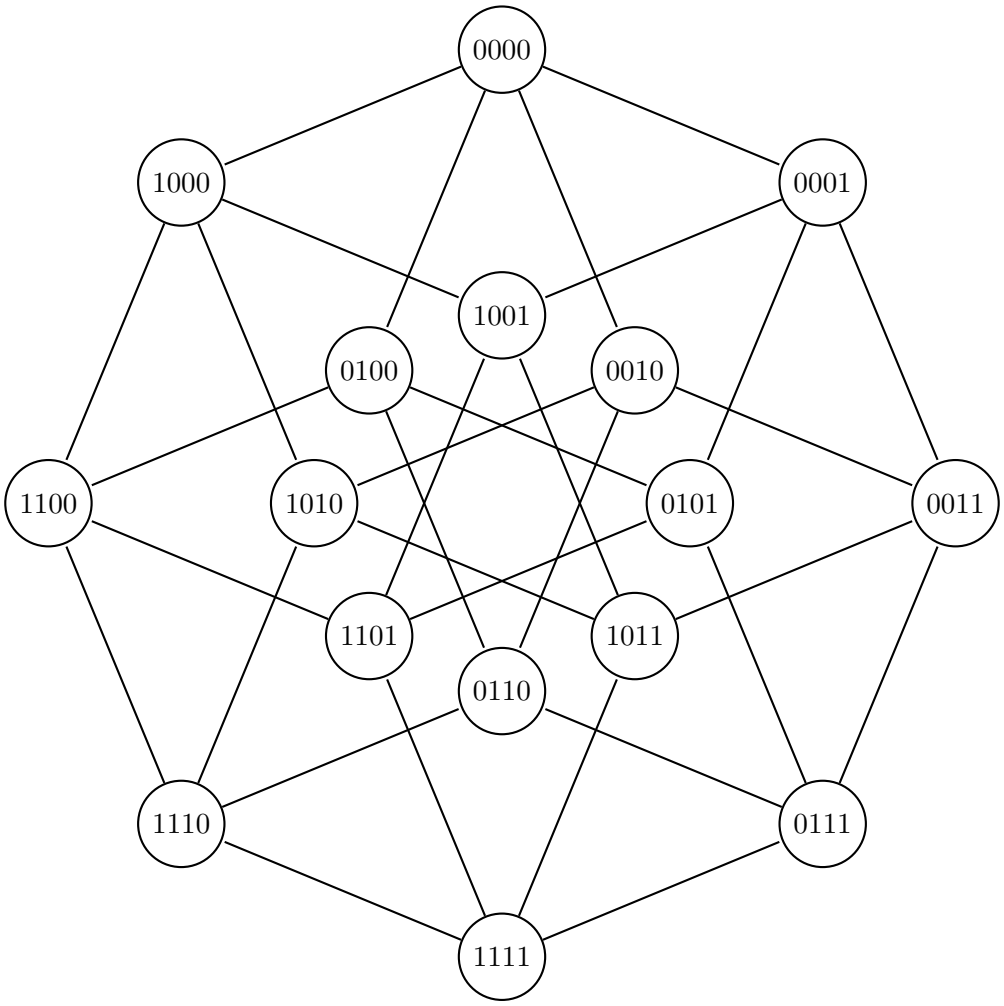

Figure 7.1: The 4-dimensional hypercube graph

We let $\mathcal{H}^{\pm} \subset \mathcal{H}$ and $\widehat{\mathcal{H}^{\pm}} \subset \widehat{\mathcal{H}}$ denote the subclass of Boolean functions in the evaluation and Fourier domains respectively.

We define the ℓ_2 **norm** of $f \in \mathcal{H}$ to be $\|f\|_2 \stackrel{\text{def}}{=} \sqrt{\mathbb{E}_x[f(x)^2]}$ and the **relative Hamming norm** to be $\|f\|_0 = \Pr_x[f(x) \neq 0]$. By **Plancherel's Theorem**, we have that

$$\|f\|_2 = \sqrt{\sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)^2}.$$

For $f \in \mathcal{H}^{\pm}$, we have $\|f\|_2 = 1$. We also define, for $f, g \in$ the ℓ_2 **distance** $\mathrm{dist}_2(f, g) = \|f - g\|_2$,

and the **relative Hamming distance** $\mathrm{dist}_0(f,g) = \|f-g\|_0$, that is, the probability that $f(x) \neq g(x)$ for $x \in \mathbb{F}_2^n$ chosen uniformly at random. This definition of dist_0 extends to arbitrary functions over a common finite domain.

If $f, g \in \mathcal{H}^{\pm}$, then $|f-g|$ only takes the values 0 and 2, such that their ℓ_2 and relative Hamming distances are related by the following:

Fact 7.1.1. *Let $f, g \in \mathcal{H}^{\pm}$, and let $\mu = \mathrm{dist}_0(f,g)$ and $\epsilon = \mathrm{dist}_2(f,g)$. Then $\epsilon = 2\sqrt{\mu}$, or equivalently $\mu = \epsilon^2/4$.*

Given a class of functions $\mathcal{C} \subset \mathcal{H}$, we let

$$\mathrm{dist}_k(f, \mathcal{C}) \stackrel{\mathrm{def}}{=} \inf_{g \in \mathcal{C}} \mathrm{dist}_k(f,g) \quad \text{for} \quad k = 0, 2.$$

We let $\mathcal{H}_t \subset \mathcal{H}$ denote the family of t-sparse functions, i.e., those functions with at most t nonzero Fourier coefficients, and $\mathcal{H}_t^{\pm}$ denote the subclass of t-sparse Boolean functions. We say $f \in \mathcal{H}^{\pm}$ is ϵ-**close** to t-sparse if $\mathrm{dist}_2(f, \mathcal{H}_t) \leq \epsilon$, otherwise we say f is ϵ-**far** from t-sparse. We define sparsity testers that distinguish t-sparse Boolean functions from those that are distant from t-sparse functions under either ℓ_2 or relative Hamming distance.

Definition 7.1.2. *An ϵ-ℓ_2 t-**sparsity tester** for $f \in \mathcal{H}^{\pm}$ is an algorithm $\mathcal{A}$ that takes ϵ, f and t and such that, with probability at least $2/3$, the following hold:*

- *If f is t-sparse, then $\mathcal{A}$ accepts;*
- *If f is ϵ-far from t-sparse, then $\mathcal{A}$ rejects.*

*A μ-**Hamming** t-**sparsity tester** for $f \in \mathcal{H}^{\pm}$ is an algorithm $\mathcal{A}$ that takes f and t, and such that, with probability at least $2/3$, the following hold:*

- *If f is t-sparse, then $\mathcal{A}$ accepts;*
- *If $\mathrm{dist}_0(f, \mathcal{H}_t^{\pm}) > \mu$, then $\mathcal{A}$ rejects.*

We will also consider *vector-valued* analogues of classes of functions. For instance, we consider functions of the form $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{C}^m$ for $m \geq 1$. Here the Fourier transform $\widehat{\boldsymbol{f}}$ of $\boldsymbol{f}$ is merely the vector-valued function whose components are the Fourier transforms of the components of $\boldsymbol{f}$. We extend the notion of support in the natural way, letting $\mathrm{supp}(f) = \{\alpha \in \mathbb{F}_2^n : \widehat{\boldsymbol{f}}(\alpha) \neq \mathbf{0} \in \mathbb{C}^m\}$.

We write $\vec{\mathcal{H}}$ to denote the family of vector functions over the hypercube, and use $\vec{\mathcal{H}}, \vec{\mathcal{H}}_t$ and $\vec{\mathcal{H}}_t^{\pm}$ similarly. When n or both n and m are not clear from the context we may write $\mathcal{H}(n)$, $\vec{\mathcal{H}}(n)$, $\vec{\mathcal{H}}(n,m)$, and similarly for $\mathcal{H}^{\pm}, \mathcal{H}_t, \mathcal{H}_t^{\pm}$, and their vector-valued variants.

7.1.1 Previous work

Fourier sparsity testing was first studied by Gopolan et al. in [Gop+11]. There the authors give an algorithm based on **projections** of f (see Section 7.1.2). They gave an ϵ-ℓ_2 *t-sparsity-tester* for the class $\mathcal{H}_t^{\pm} \subset \mathcal{H}^{\pm}$ with query complexity $\mathcal{O}(t^6 \log t/\epsilon^2)$. They achieve this result by "hashing" the Fourier terms of f into $\mathcal{O}(t^2)$ bins, such that with high probability the t significant Fourier coefficients of f land in unique bins. They then probabilistically estimate the Fourier mass concentrated in each of the $\mathcal{O}(t^2)$ bins up to an additive factor $\mathcal{O}(\mu^2/t^2)$. Section 7.1.2 discusses this hashing and estimation.

They show, moreover, that if f is a Boolean function that is close to the class of t-sparse functions $\mathcal{H}_t$, then it is close to the class of Boolean t-sparse functions $\mathcal{H}_t^{\pm}$. Namely, they showed the following.

Theorem 7.1.3 (Thm. 3.4 [Gop+11])**.** *Let $f \in \mathcal{H}^{\pm}$ be such that $\mathrm{dist}_2(f, \mathcal{H}_t) \leq \epsilon$, where $\epsilon \leq \frac{1}{20t^2}$. Then $\mathrm{dist}_2(f, \mathcal{H}_t^{\pm}) \leq 2\epsilon$, or equivalently, $\mathrm{dist}_0(f, \mathcal{H}_t^{\pm}) \leq \epsilon^2/2$.*

By setting $\epsilon = \min(2\sqrt{\mu}, \frac{1}{20t^2})$, the ϵ-ℓ_2 t-sparsity tester of Gopolan et al. becomes a μ-Hamming t-sparsity tester with query-complexity $\mathcal{O}(t^6 \log t/\mu^2 + t^{14} \log t)$. This same approach may be used to develop a μ-Hamming t-sparsity tester from any ϵ-ℓ_2 t-sparsity tester.

In [WY13], Wimmer and Yoshida further analyze the technique of Gopolan et al. to get a tolerant tester that accepts if $\mathrm{dist}_2(f, \mathcal{H}_t) < \epsilon/3$ and rejects if $\mathrm{dist}_2(f, \mathcal{H}_t) > \epsilon$, each with probability $2/3$, and with query complexity $\mathrm{poly}(t/\epsilon)$. This method allows one to estimate distance from t-sparsity up to a multiplicative factor. In [HL13a], Hatami and Lovett give a method of estimating $\mathrm{dist}_2(f, \mathcal{H}_t)$ up to an *additive* factor, albeit with a query-complexity that is a tower of exponents of height t.

7.1.2 Projections and restrictions

In this section we introduce projection and restriction operations on $f \in \mathcal{H}$, which we use in sparse Fourier testing and transform algorithms. We let $V^{\top}$ denote the orthogonal space of a subspace $V \leq \mathbb{F}_2^n$. For a choice of $v_1, \ldots, v_r \in \mathbb{F}_2^n$, we let $\boldsymbol{v}$ be the $n \times r$ matrix whose ith column is given by v_i, and identity $\boldsymbol{v}$ with its column space $V = \mathrm{span}(\{v_1, \ldots, v_r\})$. For $\beta \in \mathbb{F}_2^n$, we define the **Fourier projection** of f onto the coset $\beta + V^{\top}$ by

$$\widehat{f_{\beta}^{\boldsymbol{v}}}(\alpha) = \begin{cases} \widehat{f}(\alpha) & \text{if } \alpha \in \beta + V^{\top}, \\ 0 & \text{otherwise.} \end{cases}$$

We will often simply call $f_{\alpha}^{\boldsymbol{v}}$ a **projection** of f. One can estimate an evaluation or the Fourier mass of a projection function by way of the following fact.

Fact 7.1.4 ([Gop+11])**.**

$$f_\alpha^{\boldsymbol{v}}(x) = \mathbb{E}_{y\in V}[f(x+y)\chi_\alpha(y)], \qquad \textit{and} \qquad \|f_\alpha^{\boldsymbol{v}}\|_2^2 = \mathbb{E}_{x\in\mathbb{F}_2^n}\left[\left(\mathbb{E}_{y\in V}[f(x+y)\chi_\alpha(y)]\right)^2\right].$$

Projections allow us to hash Fourier coefficients α to "bins" corresponding to cosets of V. For $\kappa \in \mathbb{F}_2^r$, we let the bin $\mathcal{B}(\kappa)$ denote the set of $\alpha \in \mathbb{F}_2^n$ such that $\boldsymbol{v}^\top \alpha = \kappa$. That is,

$$\mathcal{B}(\kappa) \stackrel{\text{def}}{=} \{\alpha \in \mathbb{F}_2^n : v_i^\top \alpha = \kappa_i \text{ for } i \in [s]\}.$$

If the v_i form a linearly independent set, then the bins $\mathcal{B}(\kappa)$ comprise the cosets of $V^\top$. Otherwise, some of the bins $\mathcal{B}(\kappa)$ will be empty, with the cosets of $V^\top$ being the nonempty bins. The set hash functions corresponding to choices of $\boldsymbol{v} \in \mathbb{F}_2^{n\times r}$ forms a pairwise independent hash family.

Fact 7.1.5. *Choose $\boldsymbol{v} \in \mathbb{F}_2^{n\times r}$ independently and uniformly at random. For $V = \operatorname{span}(\{v_1, \ldots, v_r\})$ and fixed $\kappa \in \mathbb{F}_2^t$, let X_α be the indicator random variable for the event that $\alpha \in \mathcal{B}(\kappa)$. Then the collection of random variables $\{X_\alpha\}_{\alpha\in\mathbb{F}_2^n}$ is pairwise independent. Moreover, for $\alpha \neq \mathbf{0}$, $\mathbb{E}[X_\alpha] = 2^{-r}$.*

We also define, for $x \in \mathbb{F}_2^n$, the transformation $f|_x^{\boldsymbol{v}}$ of f by

$$f|_x^{\boldsymbol{v}} : \mathbb{F}_2^t \to \mathbb{C}, \qquad y \mapsto f(\boldsymbol{v}^\top y + x) = f(v_1^\top y + x_1, v_2^\top y + x_2, \ldots, v_n^\top y + x_n).$$

For lack of a better term, we will refer to $f|_x^{\boldsymbol{v}}$ as the **restriction** of f to $x + V$. We let $f|^{\boldsymbol{v}} = f|_{\mathbf{0}}^{\boldsymbol{v}}$. One can compute $f|_x^{\boldsymbol{v}}$ by evaluating f over the image $\boldsymbol{v}\alpha, \alpha \in \mathbb{F}_2^r$. We can compute all elements of the column space V at a cost of $\mathcal{O}(n2^r)$ bit operations on a unit-cost RAM, by producing all linear combinations of the first i columns of $\boldsymbol{v}$ for increasing values of i. Adding x to each element of V to produce $x + V$ costs $\mathcal{O}(n2^r)$ similarly, which yields the following cost.

Lemma 7.1.6. *Given $f \in \mathcal{H}(n, m)$, $\boldsymbol{v} \in \mathbb{F}_2^{n\times r}$ and $x \in \mathbb{F}_2$, one can compute $f|_x^{\boldsymbol{v}}$ with a cost of $\mathcal{O}(2^r)$ queries and an additional bit operation cost of $\mathcal{O}(nm2^r)$.*

Restrictions and projections are closely related. Observe that for any $\alpha, x \in \mathbb{F}_2^n, \beta \in V^\top, y \in \mathbb{F}_2^r$, that

$$\chi_{\alpha+\beta}(\boldsymbol{v}y + x) = \chi_{\alpha+\beta}(x)(-1)^{\alpha^\top \boldsymbol{v}y}\underbrace{(-1)^{\beta^\top \boldsymbol{v}y}}_{=1} = \chi_{\alpha+\beta}(x)\chi_{\boldsymbol{v}^\top\alpha}(y),$$

from which we obtain the following.

Lemma 7.1.7. *For $\alpha \in \mathbb{F}_2^n$, let $\kappa = V^\top\alpha \in \mathbb{F}_2^r$. Then $f_\alpha^{\boldsymbol{v}}(x) = \widehat{f|_x^{\boldsymbol{v}}}(\kappa)$.*

7.1.3 The FFT over the hypercube

Given $f \in \mathcal{H}$, we may compute $\widehat{f}$ via the Fast Fourier Transform, also known as the Fast Walsh-Hadamard Transform. By treating f and $\widehat{f}$ as vectors indexed by $\mathbb{F}_2^n$, we can describe the map from f to $\widehat{f}$ as a $2^n \times 2^n$ linear map

$$\boldsymbol{F}_n \stackrel{\text{def}}{=} 2^{-n} \left[\chi_\alpha(\beta)\right]_{\alpha,\beta\in\mathbb{F}_2^n} = 2^{-n} \begin{bmatrix} \chi_{0\cdots00}(0\cdots00) & \chi_{0\cdots00}(0\cdots01) & \cdots & \chi_{0\cdots00}(1\cdots11) \\ \chi_{0\cdots01}(0\cdots00) & \chi_{0\cdots01}(0\cdots01) & \cdots & \chi_{0\cdots01}(1\cdots11) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_{1\cdots11}(0\cdots00) & \chi_{1\cdots11}(0\cdots01) & \cdots & \chi_{1\cdots11}(1\cdots11) \end{bmatrix}.$$

The matrix $\boldsymbol{F}_n$ admits a factorization

$$\boldsymbol{F}_n = \prod_{i=1}^{n} (I_{n-i-1} \otimes \boldsymbol{F}_1 \otimes I_i),$$

where the product can be read in any order, and $\boldsymbol{F}_1 = \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$. For instance, we have that

$$\boldsymbol{F}_2 = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & -1/2 & 0 & -1/2 \end{bmatrix}.$$

This factorization describes the FFT, which we give as pseudocode in Algorithm 23. We use the binary concatenation operation $\|$.

Algorithm 23: The FFT over the hypercube

Input: $f \in \vec{\mathcal{H}}(n, m)$
Output: $\widehat{f}$

$g \leftarrow f$;
for $i \in [n]$ **do**
  for $\gamma \in \mathbb{F}_2^i$ *and* $\kappa \in \mathbb{F}_2^{n-i-1}$ **do**
    $\alpha \leftarrow \gamma\|0\|\kappa$;
    $\beta \leftarrow \gamma\|1\|\kappa$;
    $\begin{bmatrix} g(\alpha) \\ g(\beta) \end{bmatrix} \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} g(\alpha) \\ g(\beta) \end{bmatrix}$;
return $2^{-n} g$

Lemma 7.1.8. *Algorithm 23 takes $f \in \vec{\mathcal{H}}(n,m)$ and produces $\widehat{f}$. It costs $\widetilde{\mathcal{O}}(2^n)$ queries to f, and $\mathcal{O}(2^n n)$ additions and subtractions in $\mathbb{C}$. If $f \in \mathcal{H}^{\pm}(n,m)$, its bit cost is $\mathcal{O}(2^n mn^2)$ on a unit-cost RAM.*

Proof. The query cost is clear: we query f at all 2^n evaluation points in $\mathbb{F}_2^n$. The number of iterations of the inner for loop is $2^{n-1}n$, from which the arithmetic cost follows.

In the case that $f \in \mathcal{H}^{\pm}(n,m)$, each of the entries of each component of f is either 1 or -1. It follows via induction that the values of the components of $g(\alpha)$, $\alpha \in \mathbb{F}_2^n$, at the start of the ith iteration of the outer for loop, are at most $2^{i-1} \in \mathcal{O}(2^n)$ in absolute value. Thus, in this case, addition or subtraction of any two components of $g(\alpha)$ and $g(\beta)$ costs at most $\mathcal{O}(n)$ bit operations. As the indices α, β are also n bits, this brings the bit cost of one iteration of the inner for loop to $\mathcal{O}(nm)$, from which the total bit cost $\mathcal{O}(2^n mn^2)$ follows. □

As $\boldsymbol{F}_n^{-1} = 2^{-n}\boldsymbol{F}_n$, the inverse FFT follows identically up to scalar multiplication.

7.2 Sparse Fourier Transforms over the hypercube

In this section we give an efficient Sparse Fourier Transform (SFT) algorithm for f acting over the hypercube, when f is known to be t-sparse. An SFT algorithm was given by Goldreich and Levin in [GL89] for the purposes of constructing hard-core predicates for arbitrary one-way functions. Kushilevitz and Mansour describe a similar algorithm in [KM93]. Their algorithm takes arbitrary $f \in \mathcal{H}$ and produces a t-sparse approximation to f that is within a constant factor of the best t-sparse approximation to f. That is, it produces $\widehat{g}$, where g is t-sparse, such that $\mathrm{dist}_2(f,g) < (1+\epsilon)\mathrm{dist}_2(f,\mathcal{H}_t)$. Its query complexity is $\widetilde{\mathcal{O}}(nt^2 \cdot \mathrm{poly}(\epsilon^{-1}))$. In [Lev93], Levin described an idea which gives a probabilistic $\mathcal{O}(nt \cdot \mathrm{poly}(\epsilon^{-1}))$-query SFT. For completeness, we give a description and analysis of such an algorithm here.

Observe that if a Fourier index $\alpha \in \mathrm{supp}(f)$ uniquely hashes to $\mathcal{B}(\kappa)$, then

$$\widehat{f|^{\boldsymbol{v}}_{e_j}} = \widehat{f|^{\boldsymbol{v}}_x}(\kappa)(-1)^{\alpha_j},$$

where e_j is the jth column of the $n \times n$ identity matrix. From this we can recover the jth bit of α, α_j. This idea was described by Levin in [Lev93], and can be used to give a $O(nk)$-query Sparse Fourier Transform algorithm that succeeds with probability at least $2/3$.

Lemma 7.2.1. *Procedure **ExactSFT**(f,n,t) takes as input t-sparse $f \in \vec{\mathcal{H}}_t(n,m)$ and produces $\widehat{f}$ with probability at least $2/3$. Its query cost is $\mathcal{O}(mnt)$.*

If $f \in \vec{\mathcal{H}}_t^{\pm}(n,m)$, the bit cost is $\mathcal{O}(mnt)$ bit operations.

Procedure ExactSFT(f, n, t)

Input: $f \in \vec{\mathcal{H}}_t(n)$
Output: With probability at least $2/3$, $\widehat{f}$

1 $\widehat{g} \leftarrow \mathbf{0} \in \mathcal{H}(n)$;
2 $s \leftarrow t$;
3 **for** $i \leftarrow 1$ **to** $\lceil \log_4 t \rceil$ **do**
4 $\quad r \leftarrow i + \lceil \log(18(s-1)) \rceil$;
5 $\quad$ Choose $\boldsymbol{v} \in \mathbb{F}_2^{n \times r}$ uniformly at random;
6 $\quad$ Compute $\widehat{f|^{\boldsymbol{v}}}$ and $\widehat{f|^{\boldsymbol{v}}_{e_j}}$ for each $j \in [n]$ via the FFT;
7 $\quad \widehat{h|^{\boldsymbol{v}}} \leftarrow \widehat{f|^{\boldsymbol{v}}} - \widehat{g|^{\boldsymbol{v}}}$;
8 $\quad$ **for** $j \in [n]$ **do** $\widehat{h|^{\boldsymbol{v}}_{e_j}} \leftarrow \widehat{f|^{\boldsymbol{v}}_{e_j}} - \widehat{g|^{\boldsymbol{v}}_{e_j}}$;
9 $\quad \mathcal{I} \leftarrow \mathrm{supp}(\widehat{h|^{\boldsymbol{v}}}) \subset \mathbb{F}_2^r$;
10 $\quad$ **for** $\kappa \in \mathcal{I}$ **do**
11 $\quad\quad \alpha_\kappa \leftarrow \mathbf{0} \in \mathbb{F}_2^n$;
12 $\quad\quad$ **for** $j \in [n]$ **do**
13 $\quad\quad\quad$ **if** $\widehat{h|^{\boldsymbol{v}}_{e_j}}(\kappa) = \widehat{h|^{\boldsymbol{v}}}(\kappa)$ **then** $\alpha_{\kappa j} \leftarrow 0$;
14 $\quad\quad\quad$ **else if** $\widehat{h|^{\boldsymbol{v}}_{e_j}}(\kappa) = -\widehat{h|^{\boldsymbol{v}}}(\kappa)$ **then** $\alpha_{\kappa j} \leftarrow 1$;
15 $\quad\quad\quad$ **else** $\mathcal{I} \leftarrow \mathcal{I} \setminus \{\kappa\}$;
16 $\quad$ **for** $\kappa \in \mathcal{I}$ **do** $\widehat{g}(\alpha_\kappa) \leftarrow \widehat{f|^{\boldsymbol{v}}}(\kappa)$;
17 $\quad s \leftarrow \lfloor s/4 \rfloor$;
18 **return** $\widehat{g}$;

Proof. Let $h = f - g$, and suppose h is s-sparse at the beginning of the ith iteration of the outer for loop. Then, for our choice of V and using the union bound, the probability that a nonzero Fourier coefficient $\widehat{h}(\alpha)$ is in a collision in $h|^{\boldsymbol{v}}$ is at most $(s-1)2^{-r} \leq 2^{-i}/3$. Let X_i be the indicator event that more than $s/6$ nonzero Fourier coefficients collide in the ith iteration. By linearity of expectation and Markov's inequality, $\mathbb{E}[X_i] < 2^{-1}/18$. By the union bound, the probability of this event occuring for any iteration of the outer for loop is at most $(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots)/3 < 1/3$.

Suppose then that this does not occur. It follows from the discussion preceding Lemma 7.2.1 that the algorithm correctly constructs the index α of every nonzero, non-colliding Fourier coefficient $\widehat{h}(\alpha)$. For every bucket $\mathcal{B}(\kappa)$ where a collision occured, either an erroneous Fourier coefficient is added to g, or κ is removed from $\mathcal{I}$ on line 15, where it is detected that the bin $\mathcal{B}(\kappa)$ is a multiple. Thus, the algorithm constructs at least $5s/6$ singleton Fourier terms of h, and at most $s/12$ erroneous terms corresponding to the at-most $\frac{1}{2}s/6$ bins $\mathcal{B}(\kappa)$ containing collisions. Thus, after updating g on line 16 we will have that $\#h \leq s/6 + s/12 = s/4$.

We now analyze the cost. In the ith iteration the query complexity of constructing each restriction of f via the FFT is $2^r \in \mathcal{O}(s) \subseteq \mathcal{O}(2^{-i}t)$. Thus the total query complexity is $\mathcal{O}(2^{-i}nt)$. Accounting for all $\log t$ iterations gives a cost of $\mathcal{O}(mnt)$ queries.

If $f \in \mathcal{H}_t^{\pm}(n,m)$, computing each restriction $f|_e^{\boldsymbol{v}}$ costs $\mathcal{O}(mn2^r) \subseteq (nms)$ bit operations via Lemma 7.1.6. The cost of computing $\widehat{f|_e^{\boldsymbol{v}}}$ from $f|_e^{\boldsymbol{v}}$ via the FFT is $\mathcal{O}(m2^r r^2) \subseteq \mathcal{O}(ms\log^2 s)$ bit operations. Accounting for all $(n+1)$ restrictions per every iteration the outer for loop yields a total bit cost of $\mathcal{O}(mt\log^2 t + n^2 mt) = \mathcal{O}(mn^2 t)$. □

7.3 Fourier sparsity testing via homomorphism testing

In this section we restrict our attention to the sparsity testing of Boolean functions. We present a Fourier sparsity tester that generalizes the homomorphism test of Blum, Luby, and Rubinfeld, i.e., the **BLR linearity test**. Our tester takes as input a Boolean-valued function f and, with probability at least $2/3$, distinguishes between the cases that f is t-sparse and the case that f is ϵ-far from t-sparse, by which we mean $\mathrm{dist}_2(f, \mathcal{H}_t^{\pm}) > \epsilon$.

We present two versions of our tester, a simple sparsity tester, with query complexity $\mathcal{O}(t^2\epsilon^{-2})$, and a faster sparsity tester, that uses the Sparse Fourier Transform (SFT) as a subroutine, and admits a query complexity of $\mathcal{O}(t\log t\epsilon^{-2} + \epsilon^{-4})$.

7.3.1 A simple sparsity tester

Our sparsity tester algorithms rely on the selection of a matrix $\boldsymbol{v}$ which gives a hash function such that each $\alpha \in \mathrm{supp}(f)$ hashes to a unique bin.

Definition 7.3.1. *We say $\boldsymbol{v}$ **separates** $\widehat{f}$ if $\alpha - \beta \notin V^\top$, for all $\alpha \neq \beta \in \mathrm{supp}(f)$.*

In other words, $\boldsymbol{v}$ separates $\widehat{f}$ if each nonzero Fourier coefficient of f lies in a distinct bin $\mathcal{B}(\kappa)$. By Fact 7.1.5 and the union bound, we obtain the following:

Lemma 7.3.2. *Fix $\mu \in (0,1)$ and let $r = \lceil \log\big(t(t-1)/(2\mu)\big) \rceil$. Then, if $f \in \mathcal{H}_t(n)$, and $\boldsymbol{v} \in \mathbb{F}_2^{n\times r}$ is chosen uniformly at random, then $\boldsymbol{v}$ separates $\widehat{f}$ with probability at least $1-\mu$.*

If $\boldsymbol{v}$ separates $\widehat{f}$, then each projection function $f_\alpha^{\boldsymbol{v}}$ is either identically zero or the product of a character function and a nonzero constant. This will allow us to reduce sparsity testing to homomorphism testing.

Definition 7.3.3. *For $f \in \mathbb{F}_2^n$ and $\boldsymbol{v} \in \mathbb{F}_2^{n\times r}$, we define $\phi^{f,\boldsymbol{v}} : \mathbb{F}_2^n \to \widehat{\mathcal{H}}(r)$ by setting*

$$\phi^{f,\boldsymbol{v}}(x) \stackrel{\mathrm{def}}{=} \widehat{f|_x^{\boldsymbol{v}}}.$$

We compute $\boldsymbol{\phi}^{f,\boldsymbol{v}}(x)$ by computing the restriction $f|_x^{\boldsymbol{v}}$, and then computing its Fourier transform via the FFT.

Definition 7.3.4. *Let $\mathbb{C}^*$ be the multiplicative group of nonzero complex numbers and $\mathcal{I}$ be a nonempty set. The function $\boldsymbol{\phi} : \mathbb{F}_2^n \to \mathbb{C}^{\mathcal{I}}$ if an* **affine homomorphism** *if it is the componentwise product of a homomorphism $\boldsymbol{\Phi} : \mathbb{F}_2^n \to (\mathbb{C}^*)^{\mathcal{I}}$ and the vector $\boldsymbol{\phi}(\mathbf{0}) \in \mathbb{C}^{\mathcal{I}}$. We call $\boldsymbol{\phi}(\mathbf{0})$ the* **shift** *of $\boldsymbol{\phi}$.*

We note that the only homomorphisms of the form $\Phi : \mathbb{F}_2^n \to \mathbb{C}$ are the character functions χ_α.

If $\boldsymbol{v}$ separates $\widehat{f}$, then $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ is the product of the shift $\widehat{f|^{\boldsymbol{v}}}$ and the homomorphism $\boldsymbol{\Phi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}}(r)$ whose components Φ_κ, for $\kappa \in \mathbb{F}_2^r$, are given by

$$\Phi_\kappa(x) = \begin{cases} \chi_\alpha(x) & \text{if } \kappa = \boldsymbol{v}\alpha \text{ for some } \alpha \in \operatorname{supp}(\widehat{f}), \\ 1 & \text{otherwise.} \end{cases}$$

We state this as a lemma.

Lemma 7.3.5. *Suppose $\boldsymbol{v}$ separates $\widehat{f}$. Then $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ is an affine homomorphism.*

We will show now that, if $f|^{\boldsymbol{v}}$ is t-sparse, and $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ agrees with an affine homomorphism $\boldsymbol{\psi}$ on most inputs, then f agrees with some t-sparse function on most inputs as well.

Lemma 7.3.6. *Suppose $f|^{\boldsymbol{v}}$ is t-sparse, and $\operatorname{dist}_0(\boldsymbol{\phi}^{f,\boldsymbol{v}}, \boldsymbol{\psi}) < \mu < \frac{1}{2}$ for an affine homomorphism $\boldsymbol{\psi} : \mathbb{F}_2^n \to \mathbb{C}^{\mathbb{F}_2^r}$ with shift $\widehat{f|^{\boldsymbol{v}}}$. Then $\operatorname{dist}_0(f, g) < \mu$ for a t-sparse function g.*

Proof. Suppose $f|^{\boldsymbol{v}}$ is t-sparse and write $\operatorname{supp}(f|^{\boldsymbol{v}}) = \{\kappa_1, \ldots, \kappa_s\}$, where $s \leq t$. If $\kappa \notin \operatorname{supp}(f|^{\boldsymbol{v}})$, then ψ_κ is identically zero. For $i \in [s]$, we must have that $\psi_{\kappa_i} = \widehat{f|^{\boldsymbol{v}}}(\kappa_i)\chi_{\beta_i}$ for some $\beta_i \in \mathbb{F}_2^n$. Let $g = \sum_{i=1}^s \widehat{f|^{\boldsymbol{v}}}(\beta_i)\chi_{\beta_i}$. We will show that $\boldsymbol{\psi} = \boldsymbol{\phi}^{g,\boldsymbol{v}}$.

To this end it suffices to show that $\boldsymbol{v}^\top \beta_i = \kappa_i$ for each i. To that end suppose otherwise and choose i, j such that $v_j^\top \beta_i \neq \kappa_{ij}$. Then, for any $x \in \mathbb{F}_2^n$,

$$\psi_{\kappa_i}(x + v_j) = \widehat{f|^{\boldsymbol{v}}}(\kappa_i)\chi_{\beta_i}(x + v_j) = \widehat{f|^{\boldsymbol{v}}}(\kappa_i)\chi_{\beta_i}(x)(-1)^{v_j^\top \beta_i} = \psi_{\kappa_i}(x)(-1)^{v_j^\top \beta_i}. \tag{7.2}$$

However, as

$$\phi_{\kappa_i}^{f,\boldsymbol{v}}(x + v_j) = \widehat{f|^{\boldsymbol{v}}_{x+v_j}}(\kappa_i)\chi_{\beta_i}(x + v_j) = \sum_{\alpha : \boldsymbol{v}^\top \alpha = \kappa_i} \widehat{f}(\alpha)\chi_\alpha(x + v_j) = \phi_{\kappa_i}^{f,\boldsymbol{v}}(x)(-1)^{\kappa_{ij}} \tag{7.3}$$

$\phi_{\kappa_i}^{f,\boldsymbol{v}}$ can only agree with ψ_{κ_i} for one of x and $x + v_j$. This contradicts the condition that $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ and $\boldsymbol{\psi}$ agree on more than half of all inputs. From this we have that $\boldsymbol{v}^\top \beta_i = \kappa_i$ for all i, such that $\boldsymbol{\psi} = \boldsymbol{\phi}^{g,\boldsymbol{v}}$. It follows that if $\boldsymbol{\psi}(x) = \boldsymbol{\phi}^{f,\boldsymbol{v}}(x)$ for some x, then $f(y) = g(y)$ for all $y \in x + V$. In particular, it implies $f(x) = g(x)$. The result follows. □

Thus, in order to test the sparsity of f, we select $\boldsymbol{v}$ that probably separates $\widehat{f}$ for $f \in \mathcal{H}_t$, and then test if $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ is an affine homomorphism via a natural extension of the **Blum-Ruby-Lubinfeld (BLR) homomorphism test**, which tests whether a map Ψ between Abelian groups satisfies $\Psi(x)\Psi(y) = \Psi(x+y)$ for a selection of x and y. They showed the following theorem:

Theorem 7.3.7 ([BLR93])**.** *Let G and H be finite Abelian groups, and let $\Phi : G \to H$ be such that $\operatorname{dist}_0(\Phi, \Psi) > \mu$ for any group homomorphism Ψ. Then for $x, y \in G$ chosen uniformly at random, $\Phi(x) + \Phi(y) \neq \Phi(x+y)$ with probability at least $\frac{2}{9}\mu$.*

The only group homomorphisms $\Psi \in \mathcal{H}^{\pm}$ are the character functions. The only 1-sparse Boolean functions are the character functions and their negations. Thus 1-sparsity testing can be directly solved via the BLR test. The BLR test may be modified to give an affine homomorphism test.

Lemma 7.3.8. *Let $\mu < 1/2$. Suppose $\boldsymbol{\phi} : \mathbb{F}_2^n \to \mathbb{C}^{\mathcal{I}}$ is such that $\operatorname{dist}_0(\boldsymbol{\phi}, \boldsymbol{\psi}) > \mu$ for any affine homomorphism $\boldsymbol{\psi}$ with shift $\boldsymbol{\phi}(\mathbf{0}) \in \mathbb{C}^{\mathcal{I}}$. Then, for $x, y \in \mathbb{F}_2^n$ chosen independently and uniformly at random, at least one of the following does not hold with probability at least $\frac{2}{9}\mu$:*

(7.3.i) $|\phi_\kappa(z)| = |\phi_\kappa(\mathbf{0})|$ *for all* $\kappa \in \mathcal{I}$ *and* $z \in \{x, y, x+y\}$*;*

(7.3.ii) $\phi_\kappa(x)\phi_\kappa(y) = \phi_\kappa(\mathbf{0})\phi_\kappa(x+y)$ *for all* $\kappa \in \operatorname{supp}(\boldsymbol{\phi}(\mathbf{0}))$.

Proof. Define the map $\boldsymbol{\Phi}$ by

$$\boldsymbol{\Phi} : \mathbb{F}_2^n \to \left\{\pm 1, \pm\sqrt{-1}\right\}^{\mathcal{I}}, \quad x \mapsto \left(\kappa \mapsto \begin{cases} \phi_\kappa(x)/\phi_\kappa(\mathbf{0}) & \text{if } |\phi_\kappa(x)| = |\phi_\kappa(\mathbf{0})| \neq 0, \\ 1 & \text{if } \phi_\kappa(x) = \phi_\kappa(0) = 0, \\ \sqrt{-1} & \text{otherwise.} \end{cases}\right).$$

Observe that for any homomorphism $\Psi : \mathbb{F}_2^n \to (\mathbb{C}^*)^{\mathcal{I}}$, its components $\Psi_\kappa, \kappa \in \mathcal{I}$, necessarily map x to 1 or -1, so that if $\phi(x)$ disagrees with $\phi(0)\Psi(x)$, then $\Phi(x)$ disagrees with $\Psi(x)$. Thus $\operatorname{dist}_0(\Phi, \Psi) \geq \operatorname{dist}_0(\phi, \phi(0)\Psi) > \mu$. Thus, by Theorem 7.3.7, Φ fails a linearity test with probability at least $\frac{2}{9}\mu$. I.e., for $x, y \in \mathbb{F}_2^n$ chosen independently and uniformly at random, $\Phi(x) \cdot \Phi(y) \neq \Phi(x+y)$.

If we have both (7.3.i) and (7.3.ii) in addition, however, we must have that $\boldsymbol{\Phi}(x) \cdot \boldsymbol{\Phi}(y) = \boldsymbol{\Phi}(x+y)$, giving a contradiction. □

Equipped with this affine homomorphism test, we now can analyze the key component of our sparsity test,

Lemma 7.3.9. *Let $\boldsymbol{\phi} = \boldsymbol{\phi}^{f,\boldsymbol{v}}$, and $\epsilon \in [0, 1]$. Then the following hold for $x, y \in \mathbb{F}_2^n$ chosen uniformly at random:*

- *If f is t-sparse and $\boldsymbol{v}$ separates f, then (7.3.i) and (7.3.ii) hold;*
- *If f is ϵ-far from t-sparse, but $f|^{\boldsymbol{v}}$ is t-sparse, then at least one of (7.3.i) or (7.3.ii) does not hold, with probability at least $\epsilon^2/18$.*

Proof. If $\boldsymbol{v}$ separates f then $\boldsymbol{\phi}$ is an affine homomorphism per Lemma 7.3.5, such that (7.3.i) and (7.3.ii) necessarily hold.

If f is ϵ-far from t-sparse, then by Fact 7.1.1, for $\mu = \operatorname{dist}_0(f, \mathcal{H}_t^{\pm})$ we have $\mu \geq \epsilon^2/4$. If $f|^{\boldsymbol{v}}$ is t-sparse in addition, then by Lemma 7.3.6, $\operatorname{dist}_0(\boldsymbol{\phi}^{f,\boldsymbol{v}}, \boldsymbol{\psi}) \geq \mu$ for any weighted homomorphism $\boldsymbol{\psi}$. It follows from Lemma 7.3.8 that at least one of (7.3.i) and (7.3.ii) does not hold, with probability at least $2\mu/9 \geq \epsilon^2/18$. □

Algorithm 24 describes our Monte Carlo sparsity test.

Algorithm 24: A simple ϵ-ℓ_2 t-sparsity tester

Input: $f \in \mathcal{H}^{\pm}(n)$; $t \in \mathbb{Z}_{>0}$; $\epsilon > 0$
Result: Test accepts if $f \in \mathcal{H}^{\pm}(n)$, and rejects if $\operatorname{dist}_2(f, \mathcal{H}_t) > \epsilon$, each with probability at least $2/3$.

1 $r \leftarrow \lceil \log\left(3t(t-1)/2\right) \rceil$;
2 Select $\boldsymbol{v} \in \mathbb{F}_2^{n \times r}$ independently and uniformly at random;
3 Compute $\widehat{f|^{\boldsymbol{v}}}$ from $f|^{\boldsymbol{v}}$ via the FFT;
4 **if** *$f|^{\boldsymbol{v}}$ is not t-sparse* **then** Reject;
5 **for** $j \leftarrow 1, 2, \ldots, v = \lceil 18 \ln(3) \epsilon^{-2} \rceil$ **do**
6 Choose $x, y \in \mathbb{F}_2^n$ independently and uniformly at random;
7 Test that $\left|\widehat{f|_z^{\boldsymbol{v}}}(\kappa)\right| = \left|\widehat{f|^{\boldsymbol{v}}}(\kappa)\right|$ for all $z \in \{x, y, x+y\}$ and all $\kappa \in \mathbb{F}_2^r$;
8 Test that $\widehat{f|_x^{\boldsymbol{v}}}(\kappa)\widehat{f|_y^{\boldsymbol{v}}}(\kappa) = \widehat{f|_{x+y}^{\boldsymbol{v}}}(\kappa)\widehat{f|^{\boldsymbol{v}}}(\kappa)$;
9 **if** *the tests on lines 7 and 8 pass for all j* **then** Accept;
10 **else** Reject;

Theorem 7.3.10. *Algorithm 24 is a ϵ-ℓ_2 t-sparsity tester. Its query complexity is $\mathcal{O}(t^2\epsilon^{-2})$.*

Proof. Suppose f is t-sparse, such that $f|^{\boldsymbol{v}}$ is t-sparse. By Lemma 7.3.2, $\boldsymbol{v}$ separates $\widehat{f}$ with probability $2/3$, such that by Lemma 7.3.5, $\phi^{f,\boldsymbol{v}}$ is an affine homomorphism, and the tests on lines 7 and 8 always pass and the algorithm accepts.

If f is ϵ-far from t-sparse and the test does not reject on line 4, then at least one of the tests on lines 7 and 8 will not pass for a fixed iteration of the for loop, with probability at least

$\epsilon^2/18$. Thus the probability that these tests pass for $v = \lceil 18\ln(3)\epsilon^{-2}\rceil$ iterations is less than $(1-\epsilon^2/18)^v \le \exp(-v\epsilon^2/18) \le 1/3$.

The query complexity is $\mathcal{O}(2^r v) \subseteq \mathcal{O}(t^2\epsilon^{-2})$ □

From Theorem 7.1.3, we have the following Corollary:

Corollary 7.3.11. *There exists a μ-Hamming t-sparsity tester with query complexity $\mathcal{O}(t^2\mu + t^6)$.*

7.3.2 A $\mathcal{O}(t\log t)$-query t-sparsity tester

In this section we give a sparsity tester with query complexity softly linear in t. The key is to use **ExactSFT**, as opposed to the FFT, to compute the Fourier transforms of the restrictions $\widehat{f|_w^{\boldsymbol{v}}}$. We will compute Fourier transforms in batches using vectorized functions. This is so that we can more easily control the probability that **ExactSFT** fails. Specifically we will perform s affine homomorphism tests for some $s > 0$. We choose $x_1, \ldots, x_s, y_1, \ldots, y_s \in \mathbb{F}_2^n$, and set $z_i = x_i + y_i$ for each $i \in [s]$. We then implicitly let

$$\begin{aligned}\boldsymbol{F} &= (f|_w^{\boldsymbol{v}})_{w\in\mathcal{W}} \in \vec{\mathcal{H}}(r, 3s+1), \text{where}\\ \mathcal{W} &= \{\mathbf{0}, x_1, \ldots, x_s, y_1, \ldots, y_s, z_1, \ldots, z_s\},\end{aligned}$$

where $\boldsymbol{F}$ is implicitly encoded by the black-box f, $\boldsymbol{v}$, and $\mathcal{W}$. In order to reduce the query complexity of our sparsity tester to be subquadratic, we cannot compute all the evaluations of each $f|_w^{\boldsymbol{v}}, w \in \mathcal{W}$. Note that $\mathrm{supp}(\boldsymbol{F}) = \{\boldsymbol{v}\alpha : \alpha \in \mathrm{supp}(f)\}$, so that if f is t-sparse, then $\boldsymbol{F}$ is t-sparse.

We compute $\widehat{\boldsymbol{G}}$ as the output to **ExactSFT**$(\boldsymbol{F}, n, t; \mu)$, for an appropriate bound μ on the failure probability. In the case that $\boldsymbol{F}$ is t-sparse and **ExactSFT** succeeds, then $\boldsymbol{G}$ is exactly $\boldsymbol{F}$. We perform some checks that probabilistically guarantee that the output $\boldsymbol{G}$ is close to $\boldsymbol{F}$. First, we check that each component G_w of $\boldsymbol{G}$ is actually a Boolean function. We can do this, albeit with bit cost quadratic in t, by performing an inverse FFT on each component of $\boldsymbol{G}$. We then test that $\mathrm{dist}_0(\boldsymbol{F}, \boldsymbol{G})$ is probably small, e.g., that $\mathrm{dist}_0(\boldsymbol{F}, \boldsymbol{G}) \le \mu = \epsilon^2/32$ with high probability.

We show that if $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ is sufficiently close to a map $\boldsymbol{\psi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}^{\pm}}$, and $\boldsymbol{\psi}$ is sufficiently close to an affine homomorphism $\boldsymbol{\pi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}_t^{\pm}}$, with support of cardinality t, then f is close to a t-sparse function h. To that end, for any function $\boldsymbol{\psi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}}(r)$, and for $x \in \mathbb{F}_2^n$, we define

$$\|\boldsymbol{\psi}\|_2 \stackrel{\text{def}}{=} \sqrt{\mathbb{E}_{x\in\mathbb{F}_2^n}\left[\|\boldsymbol{\psi}(x)\|_2^2\right]}, \quad \text{where} \quad \|\boldsymbol{\psi}(x)\|_2 = \sqrt{\sum_{\kappa\in\mathbb{F}_2^r} \psi_\kappa^2(x)},$$

and observe that $\|\cdot\|_2$ is a norm in both settings, and that

$$\|\boldsymbol{\psi}\|_2^2 = 2^{-n} \sum_{x \in \mathbb{F}_2^n, \kappa \in \mathbb{F}_2^r} \psi_\kappa^2(x).$$

We will be interested in $\boldsymbol{\psi}$ defined by

$$\boldsymbol{\psi}(w) = \begin{cases} \widehat{G}_w & \text{if } w \in \mathcal{W} \\ \widehat{F}_w & \text{otherwise.} \end{cases} \tag{7.4}$$

Lemma 7.3.12. *Let $\boldsymbol{\psi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}}(r)$ be such that:*

- $\mathrm{dist}_2(\boldsymbol{\phi}^{f,\boldsymbol{v}}(x), \boldsymbol{\psi}(x)) \leq \epsilon/4$ *for all* $x \in \mathbb{F}_2^n$*;*
- $\boldsymbol{\psi}(x) \in \widehat{\mathcal{H}^{\pm}}$ *for all* $x \in \mathbb{F}_2^n$*;*
- $\mathrm{dist}_0(\boldsymbol{\psi}, \boldsymbol{\pi}) \leq \mu = \epsilon^2/64 < 1$ *for an affine homomorphism* $\boldsymbol{\pi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}_t^{\pm}}(r)$.

Then $\mathrm{dist}_2(f, h) \leq \epsilon$ *for a* t*-sparse function* h.

Proof. Note that, for $f, h \in \mathcal{H}(n)$,

$$\|f - h\|_2 = \|\boldsymbol{\phi}^{f,\boldsymbol{v}} - \boldsymbol{\phi}^{h,\boldsymbol{v}}\|_2.$$

We first show that $\|\boldsymbol{\phi}^{f,\boldsymbol{v}} - \boldsymbol{\psi}\|_2 \leq \epsilon/2$. First, $\|\boldsymbol{\phi}^{f,\boldsymbol{v}} - \boldsymbol{\psi}\|_2 \leq \epsilon/4$ as $\|\boldsymbol{\phi}^{f,\boldsymbol{v}}(x) - \boldsymbol{\psi}(x)\|_2 \leq \epsilon/4$ for all x. As $\|\boldsymbol{\psi}(x)\|_2 = 1$ for all x, and $\boldsymbol{\psi}$ and $\boldsymbol{\pi}$ must agree for some $y \in \mathbb{F}_2^n$, we must have that $\|\boldsymbol{\pi}(x)\|_2 = \|\boldsymbol{\pi}(y)\|_2 = 1$ for all $x \in \mathbb{F}_2^n$. It follows that

$$\|\boldsymbol{\psi} - \boldsymbol{\pi}\|_2 = \sqrt{\mathbb{E}_{x \in \mathbb{F}_2^n} \left[\|\boldsymbol{\psi}(x) - \boldsymbol{\pi}(x)\|_2^2\right]} \leq \sqrt{4\mu} = \epsilon/4,$$

and so $\|\boldsymbol{\phi}^{f,\boldsymbol{v}} - \boldsymbol{\pi}\|_2 \leq \epsilon/2$ by the triangle inequality.

We will now show that there exists a t-sparse function h satisfying $\|\boldsymbol{\pi} - \boldsymbol{\phi}^{h,\boldsymbol{v}}\|_2 \leq \epsilon/2$. Write $\mathrm{supp}(\boldsymbol{\pi}) = \{\kappa_1, \ldots, \kappa_s\}$, where $s \leq t$. Each nonzero component of $\boldsymbol{\pi}$ can be written as $\pi_{\kappa_i} = \pi_{\kappa_i}(\mathbf{0})\chi_{\beta_i}$ for some $\beta_i \in \mathbb{F}_2^n$.

Let $\mathcal{I} \subset [s]$ denote the set of $i \in [s]$ such that $\boldsymbol{v}^\top \beta_i \neq \kappa_i$, and let $\overline{\mathcal{I}} = [s] \setminus \mathcal{I}$. We claim that for every $i \in \mathcal{I}$, only one of the pairs $(\phi_{\kappa_i}^{f,\boldsymbol{v}}(x), \pi_{\kappa_i}(x))$ and $(\phi_{\kappa_i}^{f,\boldsymbol{v}}(x + v_j), \pi_{\kappa_i}(x + v_j))$ can agree in sign. To that end, we follow the argument of the proof of Lemma 7.3.6. Fix $i \in \mathcal{I}$, and let v_j be such that $v_j^\top \beta_i \neq \kappa_{ij}$. Then, following (7.2) and (7.3), $\pi_{\kappa_i}(x + v_j) = (-1)^{v_j^\top \beta_i} \pi_{\kappa_i}(v_j)$ and $\phi_{\kappa_i}^{f,\boldsymbol{v}}(x + v_j) = (-1)^{\kappa_{ij}} \phi_{\kappa_i}^{f,\boldsymbol{v}}(x)$, establishing the claim.

It follows that

$$\left(\phi^{f,\boldsymbol{v}}_{\kappa_i}(x) - \pi_{\kappa_i}(x)\right)^2 + \left(\phi^{f,\boldsymbol{v}}_{\kappa_i}(x+v_j) - \pi_{\kappa_i}(x+v_j)\right)^2 \geq 2\pi_{\kappa_i}(x)^2 = 2\pi_{\kappa_i}(\mathbf{0})^2,$$

so that

$$\mathbb{E}_{x\in\mathbb{F}_2^n}\left[\left(\phi^{f,\boldsymbol{v}}_{\kappa_i}(x) - \pi_{\kappa_i}(x)\right)^2\right] \geq \psi_{\kappa_i}(\mathbf{0})^2,$$

and thus

$$\sum_{i\in\mathcal{I}} \pi_{\kappa_i}(\mathbf{0})^2 \leq \mathbb{E}_{x\in\mathbb{F}_2^n}\left[\sum_{\kappa\in\mathbb{F}_2^r} \|\phi^{f,\boldsymbol{v}}_{\kappa_i} - \pi_{\kappa_i}(x)\|_2^2\right] \leq \epsilon^2/4.$$

Let $h = \sum_{i\in\overline{\mathcal{I}}} \pi_{\kappa_i}(\mathbf{0})\chi_{\beta_i}$. The function h is t-sparse, as $\mathrm{supp}(h) = \{\beta_i : i\}$. Then

$$\begin{aligned}
\|\boldsymbol{\phi}^{h,\boldsymbol{v}} - \boldsymbol{\pi}\|_2^2 &= \mathbb{E}_{x\in\mathbb{F}_2^n}\left[\sum_{\kappa\in\mathbb{F}_2^r}\left(\phi^{h,\boldsymbol{v}}_{\kappa}(x) - \pi_{\kappa}(x)\right)^2\right], \\
&= \mathbb{E}_{x\in\mathbb{F}_2^n}\left[\sum_{i\in\mathcal{I}}\left(\phi^{h,\boldsymbol{v}}_{\kappa_i}(x) - \pi_{\kappa_i}(x)\right)^2\right], \\
&= \sum_{i\in\mathcal{I}} \pi_{\kappa_i}(\mathbf{0})^2 \leq \epsilon^2/4.
\end{aligned}$$

Thus $\|\boldsymbol{\phi}^{h,\boldsymbol{v}} - \boldsymbol{\pi}\|_2 \leq \epsilon/2$, and $\|\boldsymbol{\phi}^{f,\boldsymbol{v}} - \boldsymbol{\phi}^{h,\boldsymbol{v}}\| \leq \epsilon$ by the triangle inequality, so that $\|f - h\|_2 \leq \epsilon$. □

We are now in position to give our algorithm.

Theorem 7.3.13. *Algorithm 25 is an ϵ-ℓ_2 tester. Its query complexity is $\mathcal{O}(t\log t/\epsilon^2 + \epsilon^{-4})$*

Proof. Suppose f is t-sparse. Then $\boldsymbol{v}$ separates $\widehat{f}$, and hence $\boldsymbol{\phi}^{f,\boldsymbol{v}}$ is an affine homomorphism, with probability at least $5/6$. Moreover, $\boldsymbol{F}$ is t-sparse, so that **ExactSFT** produces $\widehat{\boldsymbol{F}}$ with probability at least $5/6$. If these conditions hold, then the output $\widehat{\boldsymbol{G}}$ of **ExactSFT** satisfies $\widehat{G}_w(\kappa) = \boldsymbol{\phi}^{f,\boldsymbol{v}}(w)$, an affine homomorphism, so that the tests on lines 14 and 15 pass for all $i \in [u]$, and the test accepts.

Suppose now that f is ϵ-far from t-sparse, but $\boldsymbol{G}(w)$ is a t-sparse Boolean function for each $w \in \mathcal{W}$. Let $\boldsymbol{\psi}$ be defined as in (7.4). By Lemma 7.3.12, we must have that either:

- there exists $x \in \mathbb{F}_2^n$ such that $\mathrm{dist}_2(\boldsymbol{\phi}^{f,\boldsymbol{v}}(x), \boldsymbol{\psi}(x)) > \epsilon/4$; or
- $\mathrm{dist}_0(\boldsymbol{\psi}, \boldsymbol{\pi}) > \epsilon/64$ for every affine homomorphism of the form $\boldsymbol{\pi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}_t^{\pm}}(r)$.

Algorithm 25: A fast ϵ-ℓ_2 t-sparsity tester

Input: $f \in \mathcal{H}^{\pm}(n)$; $t \in \mathbb{Z}_{>0}$; $\epsilon > 0$

Result: Accepts if $f \in \mathcal{H}_t^{\pm}$ and rejects if f is ϵ-far from $\mathcal{H}_t^{\pm}$, each with probability at least $2/3$.

$r \leftarrow \lceil \log(3t(t-1)) \rceil$;

Select $\boldsymbol{v} \in \mathbb{F}_2^{n \times r}$ independently and uniformly at random;

$u \leftarrow \lceil 288 \ln(3) \epsilon^2 \rceil$;

Choose $x_1, \ldots, x_u, y_1, \ldots, y_u \in \mathbb{F}_2^n$ independently and uniformly at random;

$\mathcal{W} \leftarrow \{\mathbf{0}\} \cup \bigcup_{i=1}^{u} \{x_i, y_i, x_i + y_i\} \subset \mathbb{F}_2^n$;

$\boldsymbol{F} \leftarrow$ a black-box polynomial for $(f|_w^{\boldsymbol{v}})_{w \in \mathcal{W}}$;

$\widehat{\boldsymbol{G}} \leftarrow$ **ExactSFT**$(\boldsymbol{F}, r, t; 1/6)$;

Compute $\boldsymbol{G}$ from $\widehat{\boldsymbol{G}}$ via the inverse FFT;

if $G_w \notin \mathcal{H}_t^{\pm}(r)$ *for some* $w \in \mathcal{W}$ **then** Reject;

$s \leftarrow \lceil 64 \ln(3)/\epsilon^2 \rceil$;

Choose $z_1, \ldots, z_s \in \mathbb{F}_2^r$ independently and uniformly at random;

if $G_w(z_i) \neq f|_w^{\boldsymbol{v}}(z_i)$ *for some* $w \in \mathcal{W}$ *and* $i \in [s]$ **then** Reject;

for $i \in [u]$ **do**

  Test that $\left|\widehat{\boldsymbol{G}}_w(\kappa)\right| = \left|\widehat{\boldsymbol{G}}_{\mathbf{0}}(\kappa)\right|$ for all $w \in \{x_i, y_i, x_i + y_i\}$ and all $\kappa \in \mathbb{F}_2^r$;

  Test that $\widehat{\boldsymbol{G}}_{x_i}(\kappa)\widehat{\boldsymbol{G}}_{y_i}(\kappa) = \widehat{\boldsymbol{G}}_{x_i+y_i}(\kappa)\widehat{\boldsymbol{G}}_{\mathbf{0}}(\kappa)$;

if *the tests on lines 14 and 15 pass for all* $i \in [u]$ **then** Accept;

else Reject;

Suppose that $\mathrm{dist}_2(\boldsymbol{\phi}^{f,\boldsymbol{v}})(w), \boldsymbol{\psi}(w)) > \epsilon/4$ for some $w \in \mathcal{W}$. Then $\mathrm{dist}_0(f|_w^{\boldsymbol{v}}, G_w) > \epsilon^2/64$, such that $\mathrm{dist}_0(\boldsymbol{F}, \boldsymbol{G}) > \epsilon^2/4$, and the probability that $\boldsymbol{F}(z) = \boldsymbol{G}_(z)$, for all $z \in \{z_1, \ldots, z_s\} \subset \mathbb{F}_2^r$ chosen independently and uniformly at random, is less than $(1 - \epsilon^2/4)^s \leq \exp(-s\epsilon^2/64) \leq \exp(-\ln(6)) = 1/3$. We have that $\mathrm{dist}_2(\boldsymbol{\phi}^{f,\boldsymbol{v}}(x), \boldsymbol{\psi}(x)) = 0$ for $x \notin \mathcal{W}$.

Suppose then that $\mathrm{dist}_2(\boldsymbol{\phi}^{f,\boldsymbol{v}}(x), \boldsymbol{\psi}(x)) < \epsilon/4$ for all $x \in \mathbb{F}_2^n$. Suppose, moreover, that $\boldsymbol{\psi}$ is $\epsilon^2/64$-far from any affine homomorphism $\boldsymbol{\pi} : \mathbb{F}_2^n \to \widehat{\mathcal{H}_t^{\pm}}(r)$ in terms of relative Hamming distance. Then the probability that the tests on lines 14 and 15 fail for a fixed $i \in [u]$ is more than $\epsilon^2/288$, by Lemma 7.3.8. Thus the probability that this test passes for all $i \in [u]$, given that x_i and y_i are chosen independently and uniformly at random, is less than $(1-\epsilon^2/288)^u \leq \exp(-u\epsilon^2/288) \leq 1/3$. Thus, in either case, the algorithm rejects with probability at least $2/3$. This proves the probabilistic correctness of the algorithm.

Per Lemma 7.2.1, the call to **ExactSFT** costs $\mathcal{O}(urt) \subseteq \mathcal{O}(t \log t \epsilon^{-2})$ queries. The other check on line 12 costs $us \in \mathcal{O}(\epsilon^{-4})$ queries. □

By Theorem 7.1.3, we get the following Corollary:

Corollary 7.3.14. *There exists a μ-Hamming t-sparsity tester with query complexity $\mathcal{O}(t^5 \log t + \mu^{-2})$.*

References

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. “Efficient similarity search in sequence databases”. English. In: *Foundations of Data Organization and Algorithms*. Vol. 730. 1993, pp. 69–84. DOI: 10.1007/3-540-57301-1_5 (cit. on p. 1).

[AGR13] Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. “Faster Sparse Interpolation of Straight-Line Programs”. In: *Computer Algebra in Scientific Computing*. Vol. 8136. 2013. DOI: 10.1007/978-3-319-02297-0_5 (cit. on pp. 15, 52, 53, 56, 65, 66, 89).

[AGR14] Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. “Sparse Interpolation over Finite Fields via Low-order Roots of Unity”. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. 2014. DOI: 10.1145/2608628.2608671 (cit. on pp. 15, 52, 53, 56, 70).

[AGR15] Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. “Faster sparse multivariate polynomial interpolation of straight-line programs”. In: *Journal of Symbolic Computation* (2015). To appear. DOI: dx.doi.org/10.1016/j.jsc.2015.11.005 (cit. on pp. 15, 109, 132, 134).

[AH15] Amir Akbary and Kyle Hambrook. “A variant of the Bombieri-Vinogradov theorem with explicit constants and applications”. In: *Mathematics of Computation* 84.294 (2015), pp. 1901–1932. DOI: 10.1090/S0025-5718-2014-02919-0 (cit. on p. 33).

[AHU74] Alfred V Aho, J Hopcroft, and J Ullman. *The design and analysis of algorithms*. Addison-Wesley Reading, 1974 (cit. on pp. 5, 19).

[AK15] Andrew Arnold and Erich Kaltofen. “Error-Correcting Sparse Interpolation in the Chebyshev Basis”. In: 2015 (cit. on pp. 15, 140).

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. “PRIMES Is in P”. English. In: *Annals of Mathematics* 160.2 (2004), pp. 781–793. DOI: 10.4007/annals.2004.160.781 (cit. on p. 30).

[And+14] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. “Learning Sparse Polynomial Functions”. In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. 2014, pp. 500–510. URL: http://dl.acm.org/citation.cfm?id=2634074.2634111 (cit. on p. 12).

[AR14] Andrew Arnold and Daniel S. Roche. “Multivariate Sparse Interpolation Using Randomized Kronecker Substitutions”. In: 2014. DOI: 10.1145/2608628.2608674 (cit. on pp. 15, 109, 116).

[AR15] Andrew Arnold and Daniel S. Roche. “Output-Sensitive Algorithms for Sumset and Sparse Polynomial Multiplication”. In: *Proceedings of the 40th International Symposium on Symbolic and Algebraic Computation*. 2015. URL: http://arxiv.org/abs/1501.05296 (cit. on pp. 15, 84, 89, 97, 101).

[BCS10] Peter Brgisser, Michael Clausen, and Mohammad A. Shokrollahi. *Algebraic Complexity Theory*. 1st. Springer Publishing Company, Incorporated, 2010 (cit. on p. 21).

[Ber] Daniel Bernstein. *The Transposition Principle*. URL: cr.yp.to/transposition.html (visited on 12/05/2015) (cit. on p. 23).

[Bis+15] Anurag Bishnoi, Pete L. Clark, Aditya Potukuchi, and John R. Schmitt. “On zeros of a polynomial in a finite grid”. In: *Journal* (2015). preprint. URL: http://arxiv.org/abs/1508.06020 (cit. on p. 39).

[Bl+09] Markus Blser, Moritz Hardt, Richard J. Lipton, and Nisheeth K. Vishnoi. “Deterministically Testing Sparse Polynomial Identities of Unbounded Degree”. In: *Inf. Process. Lett.* 109.3 (2009), pp. 187–192. DOI: 10.1016/j.ipl.2008.09.029 (cit. on pp. 56, 61).

[Bla83] Richard E Blahut. *Theory and practice of error control codes*. Vol. 126. Addison-Wesley Reading (Ma) etc., 1983 (cit. on p. 47).

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. “Self-testing / correcting with applications to numerical problems”. In: *Journal of computer and system sciences* 47.3 (1993), pp. 549–595. DOI: 10.1016/0022-0000(93)90044-W (cit. on p. 164).

[Blu70] Leo I Bluestein. “A linear filtering approach to the computation of discrete Fourier transform”. In: *IEEE Transactions on Audio and Electroacoustics* 18.4 (1970), 451 –455. DOI: 10.1109/TAU.1970.1162132 (cit. on p. 82).

[BM74] A. Borodin and R. Moenck. “Fast Modular Transforms”. In: *J. Comput. Syst. Sci.* 8.3 (1974), pp. 366–386. DOI: 10.1016/S0022-0000(74)80029-2 (cit. on p. 20).

[BR60] R.C. Bose and D.K. Ray-Chaudhuri. “On a class of error correcting binary group codes”. In: *Information and Control* 3.1 (1960), pp. 68–79. DOI: 10.1016/S0019-9958(60)90287-4 (cit. on p. 47).

[BT88] Michael Ben-Or and Prasoon Tiwari. “A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. 1988, pp. 301–309. DOI: 10.1145/62212.62241 (cit. on pp. 42, 48).

[CH02] Richard Cole and Ramesh Hariharan. “Verifying Candidate Matches in Sparse and Wildcard Matching”. In: *Proceedings of the Thirty-fourth Annual ACM Symposium on Theory of Computing*. Full proofs available from . 2002, pp. 592–601. DOI: 10.1145/509907.509992 (cit. on pp. 89, 93, 94).

[CK91] David G. Cantor and Erich Kaltofen. “On Fast Multiplication of Polynomials over Arbitrary Algebras”. In: *Acta Inf.* 28.7 (1991), pp. 693–701. DOI: 10.1007/BF01178683 (cit. on p. 18).

[CKP12] Matthew T. Comer, Erich L. Kaltofen, and Clment Pernet. “Sparse Polynomial Interpolation and Berlekamp/Massey Algorithms That Correct Outlier Errors in Input Values”. In: *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*. 2012, pp. 138–145. DOI: 10.1145/2442829.2442852 (cit. on pp. 10, 143).

[CL13] Jean-Marc Couveignes and Reynald Lercier. “Fast construction of irreducible polynomials over finite fields”. English. In: *Israel Journal of Mathematics* 194.1 (2013), pp. 77–105. DOI: 10.1007/s11856-012-0070-8 (cit. on p. 19).

[Cla14] Pete L Clark. “The Combinatorial Nullstellenstze Revisited”. In: *The Electronic Journal of Combinatorics* 21.4 (2014), P4–15. URL: www.combinatorics.org/ojs/index.php/eljc/article/view/v21i4p15 (cit. on p. 39).

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, 2009 (cit. on p. 23).

[Cor+96] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. “On the LambertW function”. English. In: *Advances in Computational Mathematics* 5.1 (1996), pp. 329–359. DOI: 10.1007/BF02124750 (cit. on p. 32).

[CP06] Richard Crandall and Carl B Pomerance. *Prime numbers: a computational perspective*. 2nd. Vol. 182. Springer Science & Business Media, 2006 (cit. on p. 17).

[De+13] Anindya De, Piyush P. Kurur, Chandan Saha, and Ramprasad Saptharishi. “Fast Integer Multiplication Using Modular Arithmetic”. In: *SIAM Journal on Computing* 42.2 (2013), pp. 685–699. DOI: 10.1137/100811167 (cit. on p. 18).

[DL78] Richard A. Demillo and Richard J. Lipton. “A probabilistic remark on algebraic program testing”. In: *Information Processing Letters* 7.4 (1978), pp. 193–195. DOI: 10.1016/0020-0190(78)90067-4 (cit. on p. 39).

[Don06] David L Donoho. “Compressed sensing”. In: *Information Theory, IEEE Transactions on* 52.4 (2006), pp. 1289–1306. DOI: 10.1109/TIT.2006.871582 (cit. on p. 2).

[DP09] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009 (cit. on pp. 26, 27).

[DR09] Dimitar K Dimitrov and Fernando R Rafaeli. "Descartes' rule of signs for orthogonal polynomials". In: *East J. Approx.* 15.2 (2009), pp. 233–262 (cit. on p. 142).

[Erd46] Paul Erds. "On the coefficients of the cyclotomic polynomial". In: *Bull. Amer. Math. Soc.* 52.2 (1946), pp. 179–184. URL: http://projecteuclid.org/euclid.bams/1183507708 (cit. on p. 87).

[ET36] Paul Erds and Paul Turn. "On Some Sequences of Integers". In: *Journal of the London Mathematical Society* s1-11.4 (1936), pp. 261–264. DOI: 10.1112/jlms/s1-11.4.261 (cit. on p. 144).

[Fid73] Charles M. Fiduccia. "On the Algebraic Complexity of Matrix Multiplication." PhD thesis. 1973. URL: cr.yp.to/bib/1973/fiduccia-matrix.html (cit. on p. 23).

[Fre+88] Timothy S. Freeman, Gregory M. Imirzian, Erich Kaltofen, and Lakshman Yagati. "Dagwood: A system for manipulating polynomials given by straight-line programs". In: *ACM Trans. Math. Softw.* 14.3 (1988), pp. 218–240. DOI: 10.1145/44128.214376 (cit. on p. 12).

[Fr09] M. Frer. "Faster Integer Multiplication". In: *SIAM Journal on Computing* 39.3 (2009), pp. 979–1005. DOI: 10.1137/070711761 (cit. on p. 17).

[GCL92] Keith O Geddes, Stephen R Czapor, and George Labahn. *Algorithms for computer algebra*. Springer Science & Business Media, 1992 (cit. on p. 17).

[GG14] Torbjrn Granlund and the GMP development team. *The GNU multiple precision arithmetic library*. 6.0.0. visited on 2015/07/16. 2014. URL: https://gmplib.org/gmp-man-6.0.0a.pdf (cit. on p. 17).

[GJR10] Elena Grigorescu, Kyomin Jung, and Ronitt Rubinfeld. "A Local Decision Test for Sparse Polynomials". In: *Inf. Process. Lett.* 110.20 (2010), pp. 898–901. DOI: 10.1016/j.ipl.2010.07.012 (cit. on p. 49).

[GKS90] Dima Yu Grigoriev, Marek Karpinski, and Michael F Singer. "Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields". In: *SIAM Journal on Computing* 19.6 (1990), pp. 1059–1063 (cit. on p. 5).

[GL89] Oded Goldreich and Leonid A Levin. "A hard-core predicate for all one-way functions". In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM. 1989, pp. 25–32. DOI: 10.1145/73007.73010 (cit. on pp. 153, 160).

[GLL02] Mark Giesbrecht, George Labahn, and Wen-shin Lee. *On the Equivalence Between Prony's and Ben-Or's/Tiwari's Methods*. Tech. rep. Computer science technical report. CS-2002-23. University of Waterloo, 2002. URL: https://cs.uwaterloo.ca/research/tr/2002/23/CS-2002-23.pdf (cit. on p. 42).

[GLL09] Mark Giesbrecht, George Labahn, and Wen-shin Lee. “Symbolic-numeric Sparse Interpolation of Multivariate Polynomials”. In: *J. Symb. Comput.* 44.8 (2009), pp. 943–959. DOI: 10.1016/j.jsc.2008.11.003 (cit. on p. 41).

[Gop+11] Parikshit Gopalan, Ryan O’Donnell, Rocco A Servedio, Amir Shpilka, and Karl Wimmer. “Testing Fourier dimensionality and sparsity”. In: *SIAM Journal on Computing* 40.4 (2011), pp. 1075–1100. DOI: 10.1137/100785429 (cit. on pp. 157, 158).

[GR11] Mark Giesbrecht and Daniel S. Roche. “Diversification Improves Interpolation”. In: *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*. 2011, pp. 123–130. DOI: 10.1145/1993886.1993909 (cit. on pp. 39, 52, 56, 60, 63, 82).

[GS09] Sanchit Garg and ric Schost. “Interpolation of polynomials given by straight-line programs”. In: *Theor. Comput. Sci.* 410.27-29 (2009), pp. 2659–2662. DOI: 10.1016/j.tcs.2009.03.030 (cit. on pp. 52, 56–59).

[Har+08] Godfrey Harold Hardy, EM Wright, Roger Heath-Brown, and Joseph Silverman. *An Introduction to the Theory of Numbers*. 6th. Oxford University Press, 2008 (cit. on p. 33).

[Has+12] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. “Nearly Optimal Sparse Fourier Transform”. In: *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*. 2012, pp. 563–578. DOI: 10.1145/2213977.2214029 (cit. on p. 2).

[HL13a] Hamed Hatami and Shachar Lovett. “Estimating the Distance from Testable Affine-Invariant Properties”. In: *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. 2013, pp. 237–242. DOI: 10.1109/FOCS.2013.33 (cit. on p. 157).

[HL13b] Joris van der Hoeven and Grgoire Lecerf. “On the bit-complexity of sparse polynomial and series multiplication”. In: *Journal of Symbolic Computation* 50 (2013), pp. 227–254. DOI: http://dx.doi.org/10.1016/j.jsc.2012.06.004 (cit. on p. 104).

[HR99] Ming-Deh A Huang and Ashwin J Rao. “Interpolation of Sparse Multivariate Polynomials over Large Finite Fields with Applications”. In: *Journal of Algorithms* 33.2 (1999), pp. 204–228. DOI: 10.1006/jagm.1999.1045 (cit. on p. 5).

[JM10] Seyed Mohammad Mahdi Javadi and Michael Monagan. “Parallel Sparse Polynomial Interpolation over Finite Fields”. In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. 2010, pp. 160–168. DOI: 10.1145/1837210.1837233 (cit. on p. 5).

[Kal10] Erich L. Kaltofen. “Fifteen Years After DSC and WLSS2 What Parallel Computations I Do Today: Invited Lecture at PASCO 2010”. In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. 2010, pp. 10–17. DOI: 10.1145/1837210.1837213 (cit. on p. 48).

[Kal88] Erich Kaltofen. “Greatest Common Divisors of Polynomials Given by Straight-line Programs”. In: *J. ACM* 35.1 (1988), pp. 231–264. DOI: 10.1145/42267.45069 (cit. on p. 5).

[Kar95] Anatolii Alexeevich Karatsuba. “The complexity of computations”. In: *Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation* 211 (1995), pp. 169–183 (cit. on p. 17).

[KL03] Erich Kaltofen and Wen-shin Lee. “Early termination in sparse interpolation algorithms”. In: *Journal of Symbolic Computation* 36.3 (2003), pp. 365–400. DOI: 10.1016/S0747-7171(03)00088-9 (cit. on p. 49).

[KL96] E. Kaltofen and A. Lobo. “On Rank Properties of Toeplitz Matrices over Finite Fields”. In: *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*. 1996, pp. 241–249. DOI: 10.1145/236869.237081 (cit. on p. 135).

[KM93] Eyal Kushilevitz and Yishay Mansour. “Learning decision trees using the Fourier spectrum”. In: *SIAM Journal on Computing* 22.6 (1993), pp. 1331–1348 (cit. on pp. 1, 160).

[KO63] A. Karatsuba and Y. Ofman. “Multiplication of Multidigit Numbers on Automata”. In: *Soviet Physics Doklady* 7 (1963), p. 595 (cit. on p. 17).

[Koc+14] Murat Kocaoglu, Karthikeyan Shanmugam, Alexandros G Dimakis, and Adam Klivans. “Sparse Polynomial Learning and Graph Sketching”. In: *Advances in Neural Information Processing Systems 27*. 2014, pp. 3122–3130. URL: http://papers.nips.cc/paper/5426-sparse-polynomial-learning-and-graph-sketching.pdf (cit. on p. 12).

[KP14] Erich L. Kaltofen and Clment Pernet. “Sparse Polynomial Interpolation Codes and Their Decoding Beyond Half the Minimum Distance”. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. 2014, pp. 272–279. DOI: 10.1145/2608628.2608660 (cit. on pp. 143, 144, 150).

[Kro82] Leopold Kronecker. *Grundzge einer arithmetischen theorie der algebraischen grssen... von L. Kronecker*. G. Reimer, 1882 (cit. on p. 40).

[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*. ACM. 2001, pp. 216–223 (cit. on pp. 110, 111).

[KY13] Erich Kaltofen and George Yuhasz. “A fraction free Matrix Berlekamp/Massey algorithm”. In: *Linear Algebra and its Applications* 439.9 (2013), pp. 2515–2526. DOI: 10.1016/j.laa.2013.06.016 (cit. on p. 45).

[KY89] Erich Kaltofen and Lakshman Yagati. “Improved sparse multivariate polynomial interpolation algorithms”. English. In: *Symbolic and Algebraic Computation*. Vol. 358. 1989, pp. 467–474. DOI: 10.1007/3-540-51084-2_44 (cit. on p. 49).

[Lan05] Serge Lang. “Undergraduate Algebra”. In: *Springer, New York* 5.7 (2005), p. 8 (cit. on p. 38).

[Le 14] Franois Le Gall. “Powers of tensors and fast matrix multiplication”. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ACM. 2014, pp. 296–303 (cit. on p. 21).

[Lev93] Leonis A. Levin. “Randomness and Non-determinism”. In: *Journal of Symbolic Logic* 58.3 (1993), pp. 1102–1103. DOI: 10.2307/2275127 (cit. on pp. 153, 160).

[LS95] Yagati N. Lakshman and B David Saunders. “Sparse polynomial interpolation in non-standard bases”. In: *SIAM Journal on Computing* 24.2 (1995), pp. 387–397. DOI: 10.1137/S0097539792237784 (cit. on pp. 144, 146, 147).

[Mei02] Erik Meijering. “A chronology of interpolation: from ancient astronomy to modern signal and image processing”. In: *Proceedings of the IEEE* 90.3 (2002), pp. 319–342 (cit. on p. 1).

[Mil75] Gary L. Miller. “Riemann’s Hypothesis and Tests for Primality”. In: *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*. 1975, pp. 234–239. DOI: 10.1145/800116.803773 (cit. on p. 32).

[MNL10] Abdullah Mueen, Suman Nath, and Jie Liu. “Fast Approximate Correlation for Massive Time-series Data”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 2010, pp. 171–182. DOI: 10.1145/1807167.1807188 (cit. on p. 1).

[Moe73] R. T. Moenck. “Fast Computation of GCDs”. In: *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. 1973, pp. 142–151. DOI: 10.1145/800125.804045 (cit. on p. 19).

[Pan01] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Springer-Verlag New York, Inc., 2001 (cit. on p. 22).

[Pan89] Victor Pan. “On Some Computations with Dense Structured Matrices”. In: *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*. 1989, pp. 34–42. DOI: 10.1145/74540.74546 (cit. on p. 22).

[Pri82] Paul Pritchard. “Explaining the wheel sieve”. English. In: *Acta Informatica* 17.4 (1982), pp. 477–485. DOI: 10.1007/BF00264164 (cit. on pp. 28, 29).

[Rab80] Michael O Rabin. “Probabilistic algorithm for testing primality”. In: *Journal of Number Theory* 12.1 (1980), pp. 128–138. DOI: 10.1016/0022-314X(80)90084-0 (cit. on p. 32).

[Rad68] Charles Rader. “Discrete Fourier transforms when the number of data samples is prime”. In: *Proceedings of the IEEE* 56.6 (1968), pp. 1107–1108. DOI: 10.1109/PROC.1968.6477 (cit. on p. 82).

[RS60] Irving S Reed and Gustave Solomon. “Polynomial codes over certain finite fields”. In: *Journal of the society for industrial and applied mathematics* 8.2 (1960), pp. 300–304 (cit. on p. 47).

[RS62] J. Barkley Rosser and Lowell Schoenfeld. “Approximate formulas for some functions of prime numbers”. In: *Illinois J. Math.* 6.1 (1962), pp. 64–94. URL: http://projecteuclid.org/euclid.ijm/1255631807 (cit. on p. 29).

[Sar06] Tamas Sarlos. “Improved approximation algorithms for large matrices via random projections”. In: *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE. 2006, pp. 143–152. DOI: 10.1109/FOCS.2006.37 (cit. on p. 2).

[Sch+08] Uwe Schauz et al. “Algebraically solvable problems: describing polynomials as equivalent to explicit solutions”. In: *Electron. J. Combin* 15.1 (2008). URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v15i1r10 (cit. on p. 39).

[Sch71] A. Schnhage. “Schnelle Berechnung Von Kettenbruchentwicklungen”. In: *Acta Inf.* 1.2 (1971), pp. 139–144. DOI: 10.1007/BF00289520 (cit. on p. 19).

[Sch80] J. T. Schwartz. “Fast Probabilistic Algorithms for Verification of Polynomial Identities”. In: *J. ACM* 27.4 (1980), pp. 701–717. DOI: 10.1145/322217.322225 (cit. on p. 37).

[Sho94] Victor Shoup. “Fast Construction of Irreducible Polynomials over Finite Fields”. In: *J. Symb. Comput.* 17.5 (1994), pp. 371–391. DOI: 10.1006/jsco.1994.1025 (cit. on p. 19).

[Shp01] Igor E. Shparlinski. “Sparse Polynomial Approximation in Finite Fields”. In: *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*. 2001, pp. 209–215. DOI: 10.1145/380752.380803 (cit. on p. 142).

[Sma] SmartCare. URL: http://smartcare.be/home (visited on 05/06/2015) (cit. on p. 1).

[SS71] A. Schnhage and V. Strassen. “Schnelle Multiplikation groer Zahlen”. German. In: *Computing* 7.3-4 (1971), pp. 281–292. DOI: 10.1007/BF02242355 (cit. on p. 17).

[Sto05] Arne Storjohann. “The shifted number system for fast linear algebra on integer matrices”. In: *Journal of Complexity* 21.4 (2005), pp. 609–650. DOI: 10.1016/j.jco.2005.04.002 (cit. on p. 21).

[SW05] Igor Shparlinski and Arne Winterhof. “Noisy interpolation of sparse polynomials in finite fields”. English. In: *Applicable Algebra in Engineering, Communication and Computing* 16.5 (2005), pp. 307–317. DOI: 10.1007/s00200-005-0180-1 (cit. on p. 142).

[SY11] Shubhangi Saraf and Sergey Yekhanin. “Noisy Interpolation of Sparse Polynomials, and Applications”. In: *Proceedings of the 2011 IEEE 26th Annual Conference on Computational Complexity*. 2011, pp. 86–92. DOI: 10.1109/CCC.2011.38 (cit. on p. 142).

[Sze75] E. Szemerdi. “On sets of integers containing k elements in arithmetic progression”. English. In: *Acta Arithmetica* 27.1 (1975), pp. 199–245. URL: eudml.org/doc/205339 (cit. on p. 144).

[TY] Klaus Thull and C Yap. “A unified approach to HGCD algorithms for polynomials and integers, 1990”. Unpublished manuscript. Obtained 2015/11/23. (cit. on p. 19).

[VG03] Joachim Von Zur Gathen and Jrgen Gerhard. *Modern computer algebra*. 2nd. Cambridge university press, 2003 (cit. on pp. 17, 18, 20, 89).

[WY13] Karl Wimmer and Yuichi Yoshida. “Testing Linear-invariant Function Isomorphism”. In: *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I*. 2013, pp. 840–850. DOI: 10.1007/978-3-642-39206-1_71 (cit. on p. 157).

[Zip79] Richard Zippel. *Probabilistic algorithms for sparse polynomials*. Springer, 1979 (cit. on pp. 39, 111).

[Zip90] Richard Zippel. “Interpolating polynomials from their values”. In: *Journal of Symbolic Computation* 9.3 (1990), pp. 375–403. DOI: 10.1016/S0747-7171(08)80018-1 (cit. on pp. 38, 48, 111).