The original paraphernalia for the lottery had been lost long ago, and the black box now resting on the stool had been put into use even before Old Man Warner, the oldest man in town, was born. Mr. Summers spoke frequently to the villagers about making a new box, but no one liked to upset even as much tradition as was represented by the black box. There was a story that the present box had been made with some pieces of the box that had preceded it, the one that had been constructed when the first people settled down to make a village here. Every year, after the lottery, Mr. Summers began talking again about a new box, but every year the subject was allowed to fade off without anything's being done. The black box grew shabbier each year: by now it was no longer completely black but splintered badly along one side to show the original wood color, and in some places faded or stained.

—Shirley Jackson, *The Lottery* (1948)

# Chapter 8

# Sparsest shift interpolation

The interpolation algorithms presented in Chapter 7 produce a representation of the unknown $f \in \mathsf{R}[x]$ as a sparse polynomial in the standard power basis $1, x, x^2, \ldots$. Here we consider instead interpolation in the *shifted power basis* $1, (x-\alpha), (x-\alpha)^2, \ldots$, for some $\alpha \in \mathsf{R}$. This is useful because even polynomials with many nonzero coefficients in the standard basis may have very few terms in the shifted basis, for some very carefully chosen shift $\alpha$. The algorithm we present works over $\mathbb{Q}[x]$ and produces the *sparsest shift* $\alpha$, as well as the sparse representation in the $\alpha$-shifted power basis, in polynomial-time in the size of the output.

The basic idea of our algorithm was first presented at the MACIS 2007 conference (Giesbrecht and Roche, 2007). Significant improvements to those methods were later made and published in the journal Computational Complexity (Giesbrecht and Roche, 2010). We are indebted to Igor Shparlinski for pointing out a few very useful references on analytic number theory that we will discuss, as well as to Erich Kaltofen and Éric Schost for sharing some pre-prints and for other useful discussions on these topics.

## 8.1 Background

In the last chapter, we saw that for polynomial interpolation algorithms, the choice of representation of the output is absolutely crucial. The sparse interpolation algorithms we reviewed computed the representation of an unknown $f$ in the sparse representation, written as

$$f(x) = b_0 + b_1 x^{d_1} + b_2 x^{d_2} + \cdots + b_s x^{d_s}. \tag{8.1}$$

In this chapter, we will present a new algorithm that instead interpolates into the sparse representation in the $\alpha$-shifted power basis as in

$$f(x) = c_0 + c_1(x-\alpha)^{e_1} + c_2(x-\alpha)^{e_2} + \cdots + c_t(x-\alpha)^{e_t}. \tag{8.2}$$

With $f$ as in (8.2), we say that $\alpha$ is a *$t$-sparse shift* of $f$ because the representation has exactly $t$ non-zero and non-constant terms in this basis. When $\alpha$ is chosen so that $t$ is absolutely minimal in (8.2), we call this the *sparsest shift* of $f$.

### 8.1.1 Previous results

The most significant challenge here is computing the sparsest shift $\alpha \in \mathbb{Q}$. Computing this value from a set of evaluation points was stated as an open problem by Borodin and Tiwari (1991). Actually, their problem concerned the *decidability* of the problem for an unchosen set of evaluation points. This question is still open; all algorithms that we know of for sparse interpolation require a black box for evaluation so that the interpolated points can be chosen by the algorithm, and we do not depart from this general approach.

Grigoriev and Karpinski (1993) presented the first non-trivial algorithm to compute the sparsest shift of a polynomial. Their algorithm actually computes the sparsest shift from a black box for evaluation, but the complexity of the method is greater than the cost of dense interpolation. Therefore without loss of generality we can assume for their problem that the dense representation of $f$ is known in advance, and they show how to compute the sparsest shift $\alpha$ of $f$. The authors admit that their algorithm's dependence on the degree is probably not optimal. Our result confirms this by giving a sparsest-shift interpolation whose cost is polynomial in the logarithm of the degree.

A more efficient algorithm to compute the sparsest shift of a polynomial given in the dense representation was developed by Lakshman and Saunders (1996). Although our method is entirely different, we crucially rely on their theorem on the uniqueness and rationality of the sparsest shift (Fact 8.1 below).

Using the early interpolation version of Ben-Or and Tiwari's interpolation algorithm from Kaltofen and Lee (2003), but treating $f(x+\alpha)$ as a polynomial with indeterminate coefficients, Giesbrecht, Kaltofen, and Lee (2003) give even more efficient algorithms for computing the sparsest shift of a given polynomial.

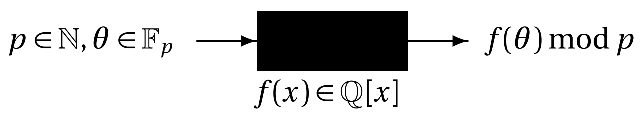
In all these cases, the size of the output in the sparsest-shifted representation could be exponentially smaller than the size of the input, represented in the standard power basis. To see that this is true, one need only polynomials such as $f(x) = (x+1)^n$. The sparsest-shifted representation size of this polynomial is proportional to $\log n$, but the size using the standard power basis — even in the sparse representation — is proportional to $n$ itself.

This motivates our problem of computing the sparsest shift directly from a black box for evaluating $f$ at chosen points, avoiding the need to ever write down $f$ in the standard power basis. Surprisingly, even though we are solving a seemingly more difficult problem, the cost of our new algorithm is competitive with the previous results just mentioned, as we will discuss later.

## 8.1.2 Problem statement and black box model

We present new algorithms to interpolate $f \in \mathbb{Q}[x]$, given a black box for evaluation, in time proportional to the size of the sparsest-shift representation corresponding to (8.2).

The black box model we use is somewhat different from those of the previous chapter:



Given a prime $p$ and an element $\theta$ in $\mathbb{F}_p$, the black box computes the value of the unknown polynomial evaluated at $\theta$ over the field $\mathbb{Z}_p$. (An error is produced exactly in those unfortunate circumstances that $p$ divides the denominator of $f(\theta)$.) We generally refer to this as a *modular black box*. To account for the reasonable possibility that the cost of black box calls depends on the size of $p$, we define $\kappa$ to be an upper bound on the number of field operations in $\mathbb{F}_p$ used in black box evaluation, for a given polynomial $f \in \mathbb{Q}[x]$. As with the remainder black box, our motivation for this definition really comes from the idea of an algebraic circuit for $f$. Given such a circuit (over $\mathbb{Q}$), we could easily construct our modular black box, and the parameter $\kappa$ would correspond to the size of the circuit (which was called $\ell$ in the previous chapter).

Some kind of extension to the standard black box, such as the modular black box proposed here, is in fact necessary, since the value of a polynomial of degree $n$ at any point other than $0, \pm 1$ will typically have $n$ bits or more. Thus, any algorithm whose complexity is proportional to $\log n$ cannot perform such an evaluation over $\mathbb{Q}$ or $\mathbb{Z}$. Other possibilities might include allowing for evaluations on the unit circle in some representation of a subfield of $\mathbb{C}$, or returning only a limited number of bits of precision for an evaluation. A similar model and approach to ours is employed by Bläser, Hardt, Lipton, and Vishnoi (2009) for deterministic identify testing of polynomials.

To be precise about our notion of size, we extend the definition of size($f$) from Chapter 6 to cover shifted-sparse polynomials. Recall that size($q$) for $q \in \mathbb{Q}$ is the number of machine words needed to represent $q$ in an IMM. So if we write $q = \frac{a}{b}$ with $a \in \mathbb{Z}$, $b \in \mathbb{N}$, and $\gcd(a, b) =$

1, and if the machine word size is $w$, then $\text{size}(q) = \lceil \log_2(|a|+1)/w \rceil + \lceil \log_2(b+1)/w \rceil + 1$. For a rational polynomial $f$ as in (8.2), define:

$$\text{size}(f) = \text{size}(\alpha) + \sum_{i=0}^{t} \text{size}(c_i) + \sum_{i=1}^{t} \text{size}(e_i). \tag{8.3}$$

The upper bound we will use for simplicity is:

$$\text{size}(f) \leq \text{size}(\alpha) + t\left(H(f) + \text{size}(n)\right), \tag{8.4}$$

where $H(f)$ is defined as $\max_{0 \leq i \leq t} \text{size}(c_i)$. Also, recall as in previous chapters that $\mathsf{M}(r)$ is the cost in ring operations of a dense degree-$r$ multiplication, and $\mathsf{N}(a)$ is the cost in word operations on an IMM of computing the product of two integers, both with absolute value at most $a$.

Our algorithms will have polynomial complexity in the smallest possible size($f$).

The remainder of this chapter is structured as follows. In Section 8.2 we show how to find the sparsest shift from evaluation points in $\mathbb{F}_p$, where $p$ is a prime with some special properties provided by some “oracle”. In Section 8.3 we show how to perform sparse interpolation given a modular black box for a polynomial. In Section 8.4 we show how to generate primes such that a sufficient number satisfy the conditions of our oracle, and discuss the effect of some number-theoretic conjectures on our algorithms. Section 8.5 provides the complexity analysis of our algorithms. We conclude in Section 8.6, and introduce some open questions.

## 8.2 Computing the Sparsest Shift

For a polynomial $f \in \mathbb{Q}[x]$, we first focus on computing the sparsest shift $\alpha \in \mathbb{Q}$ so that $f(x+\alpha)$ has a minimal number of non-zero and non-constant terms. This information will later be used to recover a representation of the unknown polynomial.

### 8.2.1 The polynomial $f^{(p)}$

Here, and for the remainder of this paper, for a prime $p$ and $f \in \mathbb{Q}[x]$, define $f^{(p)} \in \mathbb{F}_p[x]$ to be the unique polynomial with degree less than $p$ which is equivalent to $f$ modulo $x^p - x$ and with all coefficients reduced modulo $p$. Observe that this is very similar to what could be produced by the remainder black box of Chapter 7, but not quite the same. The key difference is that we simultaneously reduce the coefficients and the polynomial itself. Essentially, the remainder black box allows us to work in larger domains (when the unknown polynomial has coefficients in finite fields, for example), whereas our modular black box here allows us to work in the *smaller* domain of integers modulo $p$.

From Fermat's Little Theorem, we see immediately that $f^{(p)}(\alpha) \equiv f(\alpha) \bmod p$ for all $\alpha \in \mathbb{F}_p$. Hence $f^{(p)}$ can be found by evaluating $f$ at each point $0, 1, \ldots, p-1$ modulo $p$ and using dense interpolation over $\mathbb{F}_p[x]$.

Notice that, over $\mathbb{F}_p[x]$, $(x-\alpha)^p \equiv x-\alpha \mod x^p - x$, and therefore $(x-\alpha)^{e_i} \equiv (x-\alpha)^k$ for any $k \neq 0$ such that $e_i \equiv k \bmod (p-1)$. The smallest such $k$ is in the range $\{1,2,\ldots,p\}$; we now define this with some more notation. For $a \in \mathbb{Z}$ and positive integer $m$, define $a \operatorname{rem}_1 m$ to be the unique integer in the range $\{1,2,\ldots,m\}$ which is congruent to $a$ modulo $m$. As usual, $a \operatorname{rem} m$ denotes the unique congruent integer in the range $\{0,1,\ldots,m-1\}$.

If $f$ is as in (8.2), then by reducing term-by-term we can write

$$f^{(p)}(x) = (c_0 \operatorname{rem} p) + \sum_{i=1}^{t} (c_i \operatorname{rem} p)(x-\alpha_p)^{e_i \operatorname{rem}_1 (p-1)}, \tag{8.5}$$

where $\alpha_p$ is defined as $\alpha \operatorname{rem} p$. Hence, for some $k \le t$, $\alpha_p$ is a $k$-sparse shift for $f^{(p)}$. That is, the polynomial $f^{(p)}(x+\alpha_p)$ over $\mathbb{F}_p[x]$ has at most $t$ non-zero and non-constant terms.

Computing $f^{(p)}$ from a modular black box for $f$ is straightforward. First, use $p$ black-box calls to determine $f(i) \operatorname{rem} p$ for $i = 0,1,\ldots,p-1$. Recalling that $\kappa$ is the number of field operations in $\mathbb{F}_p$ for each black-box call, the cost of this step is $O(p\kappa \mathsf{N}(p))$ word operations. Second, we use the well-known divide-and-conquer method to interpolate $f^{(p)}$ into the dense representation (see, e.g., Borodin and Munro, 1975, Section 4.5). Since $\deg f^{(p)} < p$, this step can be performed in $O(p \log^2 p \, \mathsf{N}(p))$ word operations.

Furthermore, for any $\alpha \in \mathbb{F}_p$, the dense representation of $f^{(p)}(x+\alpha)$ can be computed in exactly the same way as the second step above, simply by shifting the indices of the already-evaluated points by $\alpha$. This immediately gives a naïve algorithm for computing the sparsest shift of $f^{(p)}$: compute $f^{(p)}(x+\gamma)$ for $\gamma = 0,1,\ldots,p-1$, and return the $\gamma$ that minimizes the number of non-zero, non-constant terms. The cost in word operations of this approach is $O(p^2 \log^2 p \, \mathsf{N}(p))$, which for our applications will often be less costly than the more sophisticated approaches of, e.g., Lakshman and Saunders (1996) or (Giesbrecht et al., 2003), precisely because $p$ will not be very much larger than $\deg f^{(p)}$.

### 8.2.2 Overview of Approach

We will make repeated use of the following fundamental theorem from Lakshman and Saunders (1996):

**Fact 8.1.** *Let* $\mathsf{F}$ *be an arbitrary field and* $f \in \mathsf{F}[x]$*, and suppose* $\alpha \in \mathsf{F}$ *is such that* $f(x+\alpha)$ *has* $t$ *non-zero and non-constant terms. If* $\deg f \ge 2t+1$ *then* $\alpha$ *is the unique sparsest shift of* $f$*.*

From this we can see that, if $\alpha$ is the unique sparsest shift of $f$, then $\alpha_p = \alpha \operatorname{rem} p$ is the unique sparsest shift of $f^{(p)}$ *provided that* $\deg f^{(p)} \ge 2t+1$. This observation provides the basis for our algorithm.

The input to the algorithms will be a modular black box for evaluating a rational polynomial, as described above, and bounds on the maximal size of the unknown polynomial. Note that such bounds are a necessity in any type of black-box interpolation algorithm, since otherwise we could never be sure that the computed polynomial is really equal to the black-box function at *every* point. Specifically, we require $B_A, B_T, B_H, B_N \in \mathbb{N}$ such that

$$\begin{aligned} \text{size}(\alpha) &\le B_A, \\ t &\le B_T, \\ \text{size}(c_i) &\le B_H, \quad \text{for } 0 \le i \le t, \\ \text{size}(n) &\le B_N. \end{aligned}$$

By considering the following polynomial:

$$c(x-\alpha)^n + (x-\alpha)^{n-1} + \cdots + (x-\alpha)^{n-t+1},$$

we see that these bounds are independent — that is, none is polynomially-bounded by the others — and therefore are all necessary.

We are now ready to present the algorithm for computing the sparsest shift $\alpha$ almost in its entirety. The only part of the algorithm left unspecified is an *oracle* which, based on the values of the bounds, produces primes to use. We want primes $p$ such that $\deg f^{(p)} \ge 2t+1$, which allows us to recover one modular image of the sparsest shift $\alpha$. But since we do not know the exact value of $t$ or the degree $n$ of $f$ over $\mathbb{Q}[x]$, we define some prime $p$ to be a *good prime for sparsest shift computation* if and only if $\deg f^{(p)} \ge \min\{2B_T+1, n\}$. For the remainder of this section, "good prime" means "good prime for sparsest shift computation." Our oracle indicates when enough primes have been produced so that at least one of them is guaranteed to have been a good prime, which is necessary for the procedure to terminate. The details of how to construct such an oracle will be considered in Section 8.4.

Our algorithm for computing the sparsest shift is presented in Algorithm 8.1.

**Theorem 8.2.** *With inputs as specified, Algorithm 8.1 correctly returns a sparsest shift $\alpha$ of $f$.*

*Proof.* Let $f, B_A, B_T, B_H, B_N$ be the inputs to the algorithm, and suppose $t, \alpha$ are as specified in (8.2).

First, consider the degenerate case where $n \le 2B_T$, i.e., the bound on the sparsity of the sparsest shift is at least half the actual degree of $f$. Then, since each $f^{(p)}$ can have degree at most $n$ (regardless of the choice of $p$), the condition of Step 6 will never be true. Hence Steps 10–14 will eventually be executed. The size of coefficients over the standard power basis is bounded by $2B_TB_A + B_H$ since $\deg f \le 2B_T$, and therefore $f$ will be correctly computed on Step 5. In this case, Fact 8.1 may not apply, i.e., the sparsest shift may not be unique, but the algorithms from Giesbrecht et al. (2003) will still produce a sparsest shift of $f$.

Now suppose instead that $n \ge 2B_T+1$. The oracle eventually produces a good prime $p$, so that $\deg f^{(p)} \ge 2B_T+1$. Since $t \le B_T$ and $f^{(p)}$ has at most $t$ non-zero and non-constant terms in the $(\alpha \operatorname{rem} p)$-shifted power basis, the value computed as $\alpha_p$ on Step 8 is exactly $\alpha \operatorname{rem} p$, by Fact 8.1. The value of $P$ will also be set to $p > 1$ here, and can only increase. So the condition of Step 10 is never true. Since the numerator and denominator of $\alpha$ are both bounded above by $2^{B_A}$, we can use rational reconstruction to compute $\alpha$ once we have the image modulo $P$ for $P \ge 2\alpha^2$. Therefore, when we reach Step 15, we have enough images $\alpha_p$ to recover and return the correct value of $\alpha$. □

**Algorithm 8.1:** Computing the sparsest shift

```
Input: A modular black box for an unknown polynomial $f \in \mathbb{Q}[x]$, bounds
       $B_A, B_T, B_H, B_N \in \mathbb{N}$ as described above, and an oracle which produces primes
       and indicates when at least one good prime must have been produced
Output: A sparsest shift $\alpha$ of $f$.
1  $P \leftarrow 1$,    $\mathcal{G} \leftarrow \emptyset$
2  while size($P$) $\leq 2B_A + 1$ do
3      $p \leftarrow$ new prime from the oracle
4      Evaluate $f(i) \operatorname{rem} p$ for $i = 0, 1, \ldots, p-1$
5      Use dense interpolation to compute $f^{(p)}$
6      if $\deg f^{(p)} \geq 2B_T + 1$ then
7          Use dense interpolation to compute $f^{(p)}(x+\gamma)$ for $\gamma = 1, 2, \ldots, p-1$
8          $\alpha_p \leftarrow$ the unique sparsest shift of $f^{(p)}$
9          $P \leftarrow P \cdot p$,    $\mathcal{G} \leftarrow \mathcal{G} \bigcup \{p\}$
10     else if $P = 1$ and oracle indicates $\geq 1$ good prime has been produced then
11         $q \leftarrow$ least prime such that size($q$) $> 2B_T B_A + B_H$ (computed directly)
12         Evaluate $f(i) \operatorname{rem} q$ for $i = 0, 1, \ldots, 2B_T$
13         Compute $f \in \mathbb{Q}[x]$ with $\deg f \leq 2B_T$ by dense interpolation in $\mathbb{F}_q[x]$ followed by
           rational reconstruction on the coefficients
14         return A sparsest shift $\alpha$ computed by a univariate algorithm from Giesbrecht
           et al. (2003) on input $f$
15 return The unique $\alpha = a/b \in \mathbb{Q}$ such that $|a|, b \leq 2^{B_A}$ and $a \equiv b\alpha_p \bmod p$ for each $p \in \mathcal{G}$,
   using Chinese remaindering and rational reconstruction
```

We still need to specify which algorithm to use to compute the sparsest shift of a densely-represented $f \in \mathbb{Q}[x]$ on Step 14. To make Algorithm 8.1 completely deterministic, we should use the univariate symbolic algorithm from Giesbrecht et al. (2003, Section 3.1), although this will have very high complexity. Using a probabilistic algorithm instead gives the following, which follows directly from the referenced work.

**Theorem 8.3.** *If the "two projections" algorithm of Giesbrecht et al. (2003, Section 3.3) is used on Step 14, then Steps 10–14 of Algorithm 8.1 can be performed with*

$$O(B_T^2 \mathsf{M}(B_T^4 B_A + B_T^3 B_H))$$

*word operations, plus $O(\kappa B_T \mathsf{M}(B_T B_A + B_H))$ word operations for the black-box evaluations.*

The precise complexity analysis proving that the entire Algorithm 8.1 has cost polynomial in the bounds given depends heavily on the size and number of primes $p$ that are used, and so must be postponed until Section 8.5.1, after our discussion on choosing primes.

**Example 8.4.** *Suppose we are given a modular black box for the following unknown polynomial:*

$$\begin{aligned} f(x) = x^{15} - 45x^{14} + 945x^{13} - 12285x^{12} + 110565x^{11} - 729729x^{10} \\ + 3648645x^9 - 14073345x^8 + 42220035x^7 - 98513415x^6 + \\ 177324145x^5 - 241805625x^4 + 241805475x^3 - 167403375x^2 \\ + 71743725x - 14348421, \end{aligned}$$

*along with the bounds $B_A = 4$, $B_T = 2$, $B_H = 4$, and $B_N = 4$. Under the simplistic assumption of single bit-length words, i.e., $w = 1$, one may easily confirm that $f(x) = (x-3)^{15} - 2(x-3)^5$, and hence these bounds are actually tight.*

*Now suppose the oracle produces $p = 7$ in Step 3. We use the black box to compute each $f(0), f(1), \ldots, f(6)$ in $\mathbb{F}_7$, and dense interpolation to compute*

$$f^{(7)}(x) = 5x^5 + 2x^4 + 3x^3 + 6x^2 + x + 4.$$

*Since $\deg f^{(7)} = 5 \geq 2B_T + 1$, we move on to Step 8 and compute each $f^{(7)}(x+\gamma)$ with $\gamma = 1, 2, \ldots, 6$. Examining these, we see that $f^{(7)}(x+3) = 5x^5 + x^3$ has the fewest non-zero and non-constant terms, and so set $\alpha_7$ to 3 on Step 8. This means the sparsest shift must be congruent to 3 modulo 7. This provides a single modular image for use in Chinese remaindering and rational reconstruction on Step 15, after enough successful iterations for different primes $p$.*

### 8.2.3 Conditions for Success

We have seen that, provided $\deg f > 2B_T$, a good prime $p$ is one such that $\deg f^{(p)} > 2B_T$. The following theorem provides (quite loose) sufficient conditions on $p$ to satisfy this requirement.

**Theorem 8.5.** *Let $f \in \mathbb{Q}[x]$ as in (8.2) and $B_T \in \mathbb{N}$ such that $t \leq B_T$. Then, for some prime $p$, the degree of $f^{(p)}$ is greater than $2B_T$ whenever the following hold:*

- $c_t \not\equiv 0 \mod p$;
- $\forall i \in \{1, 2, \ldots, t-1\}$, $e_t \not\equiv e_i \mod (p-1)$;
- $\forall i \in \{1, \ldots, 2B_T\}$, $e_t \not\equiv i \mod (p-1)$.

*Proof.* The first condition guarantees that the last term of $f^{(p)}(x)$ as in (8.5) does not vanish. We also know that there is no other term with the same degree from the second condition. Finally, the third condition tells us that the degree of the last term will be greater than $2B_T$. Hence the degree of $f^{(p)}$ is greater than $2B_T$. □

For purposes of computation it will be convenient to simplify the above conditions to two non-divisibility requirements, on $p$ and $p-1$ respectively:

**Corollary 8.6.** *Let $f, B_T, B_H, B_N$ be as in the input to Algorithm 8.1 with* $\deg f > 2B_T$. *Then there exist $C_1, C_2 \in \mathbb{N}$ with* $\text{size}(C_1) \leq 2B_H$ *and* $\text{size}(C_2) \leq B_N(3B_T - 1)$ *such that* $\deg f^{(p)} > 2B_T$ *whenever $p \nmid C_1$ and $(p-1) \nmid C_2$.*

*Proof.* Write $f$ as in (8.2). We will use the sufficient conditions given in Theorem 8.5. Write $|c_t| = a/b$ for $a, b \in \mathbb{N}$ relatively prime. In order for $c_t \operatorname{rem} p$ to be well-defined and not zero, neither $a$ nor $b$ can vanish modulo $p$. This is true whenever $p \nmid ab$. Set $C_1 = ab$. Since $\text{size}(a), \text{size}(b) \leq B_H$, $\text{size}(C_1) = \text{size}(ab) \leq 2B_H$.

Now write

$$C_2 = \prod_{i=1}^{t-1} (e_t - e_i) \cdot \prod_{i=1}^{2B_T} (e_t - i).$$

We can see that the second and third conditions of Theorem 8.5 are satisfied whenever $(p-1) \nmid C_2$. Now, since each integer $e_i$ is distinct and positive, and $e_t$ is the greatest of these, each $(e_t - e_i)$ is a positive integer less than $e_t$. Similarly, since $e_t = \deg f > 2B_T$, each $(e_t - i)$ in the second product is also a positive integer less than $e_t$. Therefore, using the fact that $t \leq B_T$, we see $C_2 \leq e_t^{3B_T - 1}$. Furthermore, $\text{size}(e_t) \leq B_N$, so we know that $\text{size}(C_2) \leq B_N(3B_T - 1)$. □

A similar criteria for success is required in (Bläser et al., 2009), and they employ Linnik's theorem which gives a polynomial-time algorithm, with a high exponent. Linnik's theorem was also employed in (Giesbrecht and Roche, 2007) to yield a much more expensive polynomial-time algorithm for finding sparse shifts than the one presented here.

## 8.3 Interpolation

Once we know the value of the sparsest shift $\alpha$ of $f$, we can trivially construct a modular black box for the $t$-sparse polynomial $f(x+\alpha)$ using the modular black box for $f$. Therefore, for the purposes of interpolation, we can assume $\alpha = 0$, and focus only on interpolating a $t$-sparse polynomial $f \in \mathbb{Q}[x]$ given a modular black box for its evaluation.

The approach we use is very similar to the techniques of the previous chapter, except that the modular black box differs from a remainder black box in a subtle way, that the coefficients and the exponents are both reduced (although not modulo the same integer!). This complication makes our analysis much more involved here. Also, the fact that we must use very small primes means that the diversification techniques of Chapter 7 will not work.

For convenience, we restate the notation for $f$ and $f^{(p)}$, given a prime $p$:

$$f = c_0 + c_1 x^{e_1} + c_2 x^{e_2} + \cdots + c_t x^{e_t}, \tag{8.6}$$

$$f^{(p)} = (c_0 \operatorname{rem} p) + (c_1 \operatorname{rem} p) x^{e_1 \operatorname{rem}_1 (p-1)} + \cdots + (c_t \operatorname{rem} p) x^{e_t \operatorname{rem}_1 (p-1)}. \tag{8.7}$$

Again, we assume that we are given upper bounds $B_H$, $B_T$, and $B_N$ on $\max_i \operatorname{size}(c_i)$, $t$, and $\operatorname{size}(\deg f)$, respectively. We also re-introduce the notation $\tau(f)$, which as in previous sections is defined to be the number of distinct non-zero, non-constant terms in the univariate polynomial $f$.

This algorithm will again use the polynomials $f^{(p)}$ for primes $p$, but now rather than a degree condition, we need $f^{(p)}$ to have the maximal number of non-constant terms. So we define a prime $p$ to be a *good prime for interpolation* if and only if $\tau(f^{(p)}) = t$. Again, the term "good prime" refers to this kind of prime for the remainder of this section.

Now suppose we have already used modular evaluation and dense interpolation (as in Algorithm 8.1) to recover the polynomials $f^{(p)}$ for $k$ distinct good primes $p_1, \ldots, p_k$. We therefore have $k$ images of each exponent $e_i$ modulo $(p_1 - 1), \ldots, (p_k - 1)$. Write each of these polynomials as:

$$f^{(p_i)} = c_0^{(i)} + c_1^{(i)} x^{e_1^{(i)}} + \cdots + c_t^{(i)} x^{e_t^{(i)}}. \tag{8.8}$$

Note that it is *not* generally the case that $e_j^{(i)} = e_j \operatorname{rem}_1 (p_i - 1)$. Because we don't know how to associate the exponents in each polynomial $f^{(p_i)}$ with their pre-image in $\mathbb{Z}$, a simple Chinese remaindering on the exponents will not work. Possible approaches are provided by (Kaltofen, 1988), (Kaltofen, Lakshman, and Wiley, 1990) or (Avendaño, Krick, and Pacetti, 2006). However, the most suitable approach for our purposes is the clever technique of (Garg and Schost, 2009), based on ideas of Grigoriev and Karpinski (1987), as we saw in the previous chapter. We interpolate the polynomial

$$g(z) = (z - e_1)(z - e_2) \cdots (z - e_t), \tag{8.9}$$

whose coefficients are symmetric functions in the $e_i$'s. Given $f^{(p_i)}$, we have all the values of $e_j^{(i)} \operatorname{rem}_1 (p_i - 1)$ for $j = 1, \ldots, t$; we just don't know the order. But since $g$ is not dependent on the order, we can compute $g \bmod (p_i - 1)$ for $i = 1, \ldots, k$, and then find the roots of $g \in \mathbb{Z}[x]$ to determine the exponents $e_1, \ldots, e_t$. Of course, this is exactly the step that our diversification technique avoided in the last chapter, but again this is not possible in this case because the primes $p$ that we use are too small.

Once we know the exponents, we recover the coefficients from their images modulo each prime. The correct coefficient in each $f^{(p)}$ can be identified because the residues of the exponents modulo $p-1$ are unique, for each chosen prime $p$. This approach is made explicit in the following algorithm.

**Algorithm 8.2:** Sparse Polynomial Interpolation over $\mathbb{Q}[x]$

```
Input: A modular black box for unknown $f \in \mathbb{Q}[x]$, bounds $B_H$ and $B_N$ as described
       above, and an oracle which produces primes and indicates when at least one
       good prime must have been returned
Output: $f \in \mathbb{Q}[x]$ as in (8.6)
1  $Q \leftarrow 1, \quad P \leftarrow 1, \quad k \leftarrow 1, \quad t \leftarrow 0$
2  while size$(P) \le 2B_H + 1$ or size$(Q) < B_N$
3  or the oracle does not guarantee a good prime has been produced do
4      $p_k \leftarrow$ new prime from the oracle
5      Compute $f^{(p_k)}$ by black box calls and dense interpolation
6      if $\tau(f^{(p_k)}) > t$ then
7          $Q \leftarrow p_k - 1$, $P \leftarrow p_k$, $t \leftarrow \tau(f^{(p_k)})$, $p_1 \leftarrow p_k$, $f^{(p_1)} \leftarrow f^{(p_k)}$, $k \leftarrow 2$
8      else if $\tau(f^{(p_k)}) = t$ then
9          $Q \leftarrow \mathrm{lcm}(Q, p_k - 1), \quad P \leftarrow P \cdot p_k, \quad k \leftarrow k + 1$
10 for $i \in \{1, \ldots, k-1\}$ do
11     $g^{(p_i)} \leftarrow \prod_{1 \le j \le t}(z - e_j^{(i)}) \bmod p_i - 1$
12 Construct $g = a_0 + a_1 z + a_2 z^2 + \cdots + a_t z^t \in \mathbb{Z}[x]$ such that $g \equiv g^{(p_i)} \bmod p_i - 1$ for
   $1 \le i < k$, by Chinese remaindering
13 Factor $g$ as $(z - e_1)(z - e_2)\cdots(z - e_t)$ to determine $e_1, \ldots, e_t \in \mathbb{Z}$
14 for $1 \le i \le t$ do
15     for $1 \le j \le k$ do
16         Find the exponent $e_{\ell_j}^{(j)}$ of $f^{(p_j)}$ such that $e_{\ell_j}^{(j)} \equiv e_i \bmod p_j - 1$
17     Reconstruct $c_i \in \mathbb{Q}$ by Chinese remaindering from residues $c_{\ell_1}^{(1)}, \ldots, c_{\ell_k}^{(k)}$
18 Reconstruct $c_0 \in \mathbb{Q}$ by Chinese remaindering from residues $c_0^{(1)}, \ldots, c_0^{(k)}$
```

The following theorem follows from the above discussion.

**Theorem 8.7.** *Algorithm 8.2 works correctly as stated.*

Again, this algorithm runs in polynomial time in the bounds given, but we postpone the detailed complexity analysis until Section 8.5.2, after we discuss how to choose primes from the "oracle". Some small practical improvements may be gained if we use Algorithm 8.2 to interpolate $f(x+\alpha)$ after running Algorithm 8.1 to determine the sparsest shift $\alpha$, since in this case we will have a few previously-computed polynomials $f^{(p)}$. However, we do not explicitly consider this savings in our analysis, as there is not necessarily any asymptotic gain.

Now we just need to analyze the conditions for primes $p$ to be good. This is quite similar to the analysis of the sparsest shift algorithm above, so we omit many of the details here.

**Theorem 8.8.** *Let $f, B_T, B_H, B_N$ be as above. There exist $C_1, C_2 \in \mathbb{N}$ with* $\operatorname{size}(C_1) \le 2B_H B_T$ *and* $\operatorname{size}(C_2) \le \frac{1}{2} B_N B_T (B_T - 1)$ *such that $\tau(f^{(p)})$ is maximal whenever $p \nmid C_1$ and $(p-1) \nmid C_2$.*

*Proof.* Let $f$ be as in (8.6), write $|c_i| = a_i/b_i$ in lowest terms for $i = 1, \ldots, t$, and define

$$C_1 = \prod_{i=1}^{t} a_i b_i, \quad C_2 = \prod_{i=1}^{t} \prod_{j=i+1}^{t} (e_j - e_i).$$

Now suppose $p$ is a prime such that $p \nmid C_1$ and $(p-1) \nmid C_2$. From the first condition, we see that each $c_i \bmod p$ is well-defined and nonzero, and so none of the terms of $f^{(p)}$ vanish. Furthermore, from the second condition, $e_i \not\equiv e_k \bmod p-1$ for all $i \neq j$, so that none of the terms of $f^{(p)}$ collide. Therefore $f^{(p)}$ contains exactly $t$ non-constant terms. The bounds on $C_1$ and $C_2$ follow from the facts that each $\operatorname{size}(a_i), \operatorname{size}(b_i) \le B_H$ and each difference of exponents $(e_j - e_i)$ has size at most $B_N$. □

## 8.4 Generating primes

We now turn our attention to the problem of generating primes for the sparsest shift and interpolation algorithms. We first present our algorithm for generating suitable primes, and we make use of powerful results from analytic number theory to estimate its running time. However, experience suggests that the true complexity of our algorithm is even better than we can prove. We briefly examine some related mathematical problems and indications towards the true cost of our method.

### 8.4.1 Prime generation algorithm

Recall that the algorithms above for sparsest shift computation and interpolation assumed we had an "oracle" for generating good primes, and indicating when at least one good prime must have been produced. We now present an explicit and analyzed algorithm for this problem.

The definition of a "good prime" is not the same for the algorithms in Section 8.2 and Section 8.3. However, Corollary 8.6 and Theorem 8.8 provide a unified presentation of sufficient conditions for primes being "good". Here we call a prime which satisfies those sufficient conditions a *useful prime.* So every useful prime is good (with the bounds appropriately specified for the relevant algorithm), but some good primes might not be useful.

We first describe a set $\mathscr{P}$ of primes such that the number and density of useful primes within the set is sufficiently high. We will assume that there exist numbers $C_1, C_2$, and useful primes $p$ are those such that $p \nmid C_1$ and $(p-1) \nmid C_2$. The numbers $C_1$ and $C_2$ will be unknown, but we will assume we are given bounds $\beta_1$, $\beta_2$ such that $\log_2 C_1 \le \beta_1$ and $\log_2 C_2 \le \beta_2$. Suppose we want to find $\ell$ useful primes. We construct $\mathscr{P}$ explicitly, of a size guaranteed to contain enough useful primes, then enumerate it.

The following fact is immediate from (Mikawa, 2001), though it has been somewhat simplified here, and the use of (unknown) constants is made more explicit. This will be important in our computational methods.

For $q \in \mathbb{Z}$, let $S(q)$ be the smallest prime $p$ such that $q \mid (p-1)$.

**Fact 8.9** (Mikawa 2001)**.** *There exists a constant $\mu > 0$, such that for all $n > \mu$, and for all integers $q \in \{n, \ldots, 2n\}$ with fewer than $\mu n / \log^2 n$ exceptions, we have $S(q) < q^{1.89}$.*

Our algorithms for generating useful primes require explicit knowledge of the value of the constant $\mu$ in order to run correctly. So we will assume that we know $\mu$ in what follows. To get around the fact that we do not, we simply start by assuming that $\mu = 1$, and run any algorithm depending upon it. If the algorithm fails we simply double our estimate for $\mu$ and repeat. At most a constant number of doublings is required. We make no claim this is particularly practical.

For convenience we define

$$\Upsilon(x) = \frac{3x}{5\log x} - \frac{\mu x}{\log^2 x}.$$

**Theorem 8.10.** *Let $\log_2 C_1 \le \beta_1$, $\log_2 C_2 \le \beta_2$ and $\ell$ be as above. Let $n$ be the smallest integer such that $n > 21$, $n > \mu$ and $\Upsilon(n) > \beta_1 + \beta_2 + \ell$. Define*

$$\mathcal{Q} = \{q \text{ prime} : n \le q < 2n \text{ and } S(q) < q^{1.89}\}, \quad \mathcal{P} = \{S(q) : q \in \mathcal{Q}\}.$$

*Then the number of primes in $\mathcal{P}$ is at least $\beta_1 + \beta_2 + \ell$, and the number of useful primes in $\mathcal{P}$, such that $p \nmid C_1$ and $(p-1) \nmid C_2$, is at least $\ell$. For all $p \in \mathcal{P}$ we have $p \in O((\beta_1 + \beta_2 + \ell)^{1.89} \cdot \log^{1.89}(\beta_1 + \beta_2 + \ell))$.*

*Proof.* By (Rosser and Schoenfeld, 1962), the number of primes between $n$ and $2n$ is at least $3n/(5\log n)$ for $n \ge 21$. Applying Fact 8.9, we see $\#\mathcal{Q} \ge 3n/(5\log n) - \mu n/\log^2 n$ when $n \ge \max\{\mu, 21\}$. Now suppose $S(q_1) = S(q_2)$ for $q_1, q_2 \in \mathcal{Q}$. If $q_1 < q_2$, then $S(q_1) > q_1^2$, a contradiction with the definition of $\mathcal{Q}$. So we must have $q_1 = q_2$, and hence

$$\#\mathcal{P} = \#Q \ge \Upsilon(n) > \beta_1 + \beta_2 + \ell.$$

We know that there are at most $\log_2 C_1 \le \beta_2$ primes $p \in \mathcal{P}$ such that $p \mid C_1$. We also know that there are at most $\log_2 C_2 \le \beta_2$ primes $q \in \mathcal{Q}$ such that $q \mid C_2$, and hence at most $\log_2 C_2$ primes $p \in \mathcal{P}$ such that $p = S(q)$ and $q \mid (p-1) \mid C_1$. Thus, by construction $\mathcal{P}$ contains at most $\beta_1 + \beta_2$ primes that are not useful out of $\beta_1 + \beta_2 + \ell$ total primes.

To analyze the size of the primes in $\mathcal{P}$, we note that to make $\Upsilon(n) > \beta_1 + \beta_2 + \ell$, we have $n \in \Theta((\beta_1 + \beta_2 + \ell) \cdot \log(\beta_1 + \beta_2 + \ell))$ and each $q \in \mathcal{Q}$ satisfies $q \in O(n)$. Elements of $\mathcal{P}$ will be of magnitude at most $(2n)^{1.89}$ and hence $p \in O((\beta_1 + \beta_2 + \ell)^{1.89} \log^{1.89}(\beta_1 + \beta_2 + \ell))$. □

Given $\beta_1$, $\beta_2$ and $\ell$ as above (where $\log_2 C_1 \le \beta_1$ and $\log_2 C_2 \le \beta_2$ for unknown $C_1$ and $C_2$), we generate the primes in $\mathcal{P}$ as follows.

Start by assuming that $\mu = 1$, and compute $n$ as the smallest integer such that $\Upsilon(n) > \beta_1 + \beta_2 + \ell$, $n \ge \mu$ and $n \ge 21$. List all primes between $n$ and $(2n)^{1.89}$ using a Sieve of Eratosthenes, and store these primes in a data structure that allows constant-time membership queries. For instance, we could simply use a length-$(2n)^{1.89}$ bit array where the $i$th bit is set iff $i$ is prime. In practice, it would be more efficient to use a hash table for this purpose.

For each prime $q$ between $n$ and $2n$, determine $S(q)$, if it is less than $q^{1.89}$, by simply checking if $kq+1$ is prime for $k = 1, 2, \ldots, \lfloor q^{0.89} \rfloor$. If we find a prime $p = S(q) < q^{1.89}$, add $p$ to $\mathscr{P}$. This is repeated until $\mathscr{P}$ contains $\beta_1 + \beta_2 + \ell$ primes. If we are unable to find this number of primes, we have underestimated $\mu$ (since Theorem 8.10 guarantees their existence), so we double $\mu$ and restart the process. Obviously in practice we would not redo primality tests already performed for smaller $\mu$, so really no work need be wasted.

**Theorem 8.11.** *For* $\log_2 C_1 \le \beta_1$, $\log_2 C_2 \le \beta_2$, $\ell$, *and* $n$ *as in Theorem 8.10, we can generate* $\beta_1 + \beta_2 + \ell$ *elements of* $\mathscr{P}$ *with*

$$O((\beta_1 + \beta_2 + \ell)^{1.89} \cdot \log^{1.89}(\beta_1 + \beta_2 + \ell) \cdot \operatorname{logloglog}(\beta_1 + \beta_2 + \ell))$$

*word operations. At least* $\ell$ *of the primes in* $\mathscr{P}$ *will be useful.*

*Proof.* The method and correctness follows from the above discussion. The Sieve of Eratosthenes can be run with $O(n^{1.89} \operatorname{logloglog} n)$ bit operations (see Knuth, 1981, §4.5.4), including the construction of the bit-array as described above. Each primality test of $kq+1$ then takes constant time, and there are at most $n^{1.89}$ such tests. Since

$$n \in O((\beta_1 + \beta_2 + \ell) \cdot \log(\beta_1 + \beta_2 + \ell)),$$

the stated complexity follows. □

### 8.4.2 Using smaller primes

The analysis of our methods will be significantly improved when more is discovered about the behavior of the least prime congruent to one modulo a given prime, which we have denoted $S(q)$. This is a special case of the more general question of the least prime in an arbitrary arithmetic progression, a well-studied problem in the mathematical literature. Recall from Section 1.4 that Linnik's theorem guarantees us that $S(q) \ll q^C$ for some constant $C$. A series of results has sought explicit bounds for $C$, with the most recent progress by Xylouris (2009) giving $C \le 5.2$.

Assuming the extended Riemann hypothesis (ERH), this can be reduced unconditionally to $S(q) \ll q^2 \ln^2 q$, as reported by Bach (1990) and Heath-Brown (1992). The result of Mikawa (2001) that we employed above is even stronger, and does not require the ERH. However, recall that Mikawa's bound of $S(q) \ll q^{1.89}$ is not shown to hold for all $q$, and this is why we had to handle exceptions.

What is the true asymptotic behaviour of $S(q)$? No one knows for certain, but it seems that the answer is $S(q) \sim q \ln^2 q$. Granville and Pomerance (1990) conjecture this as an asymptotic lower bound for $S(q)$, and earlier Heath-Brown (1978) conjectured a upper bound of the same form. Today this is known as Wagstaff's conjecture, after the more general statement by Wagstaff (1979), which was also supported by an extensive (at that time) computational search, confirming the conjecture for $q < 10^6$. He also gave a heuristic argument that this should be the true asymptotic growth rate of the least prime in arithmetic progressions.

We conducted our own computational search to investigate the size of $S(q)$ and have found that $S(q) < 2q\ln^2 q$ for all primes $q$ such that $2q\ln^2 q < 2^{64}$ — that is, all $q$ for which $S(q)$ fits in a 64-bit machine word. As a point of reference, this limit is approximately $q < 10^{16}$. In fact, this search was the original motivation for some of the tricks in modular arithmetic discussed in Chapter 2. Our computation also used the result of Jaeschke (1993) which shows that only 9 Miller-Rabin trials are necessary to deterministically check primality for integers of this size.

The results of our computational search make a strong statement about the true complexity of our algorithms. On machines with at most 64-bit words (which is to say, nearly any modern computer), our algorithms cannot possibly use primes $p$ larger than $2^{64}$, since in this case the dense polynomial $f^{(p)}$ would not fit into addressable memory. (Recall the lengthy discussion of word-size in the IMM model from Chapter 1.) This means that, at least on modern hardware, the true cost of our algorithms will be given by assuming $S(q) \in O(q\log^2 q)$. In any case, the actual cost of the algorithms discussed — on any machine — will be a reflection of the true behavior of $S(q)$, even before it is completely understood by us.

Even more improvements might be possible if this rather complicated construction is abandoned altogether, as useful primes would naturally seem to be relatively plentiful. In particular, one would expect that if we randomly choose primes $p$ directly from a set which has, say, $4(\beta_1 + \beta_2 + \ell)$ primes, we might expect that the probability that $p \mid C_1$ or $(p-1) \mid C_2$ to less than, say, 1/4. Proving this directly appears to be difficult. Perhaps most germane results to this are lower bounds on the Carmichael Lambda function (which for the product of distinct primes $p_1, \ldots, p_m$ is the LCM of $p_1 - 1, \ldots, p_m - 1$), which are too weak for our purposes. See (Erdös, Pomerance, and Schmutz, 1991).

## 8.5 Complexity analysis

We are now ready to give a formal complexity analysis for the algorithms presented in Section 8.2 and Section 8.3. For all algorithms, the complexity is polynomial in the four bounds $B_A$, $B_T$, $B_H$, and $B_N$ defined in Section 8.2.2, as well as the word-size $w$ in the IMM model. Since these are each bounded above by size($f$), our algorithms will have polynomial complexity in the size of the output if these bounds are sufficiently tight.

It may be surprising to see the word-size $w$ appear in the complexity, so we briefly give an intuitive explanation for this phenomenon. Recall that the bounds $B_A$, $B_H$, and $B_N$ above are bounds on the number of words used to store the relevant values in the output. And our algorithm analysis of course will count word operations. Because of this, doing integer computations on very small integers much smaller than $2^w$ is wasteful in some sense. That is, in order to get the best complexity in word operations, we should always make sure to get the most computational power out of those word operations that we can.

Now observe that our algorithms will *definitely* be wasteful in this sense, because of the primes $p$ that drive the cost of the whole algorithm. We are doing many computations in the field $\mathbb{F}_p$, and so to maximize the value of word operations, we would normally want every $p$ to be close to the maximum value that will fit in a machine word, $2^w$. However, the cost of our algorithm depends not only on the size of $p$, but on the *value* of $p$ as well, since the algorithm repeatedly generates dense polynomials with degrees bounded by $p$. So the $p$s must be chosen as small as possible to avoid making the whole algorithm intractable. This inevitably means that computations modulo $p$ will be wasteful, and the parameter $w$ that appears in the complexity measures indicates the degree of this wastefulness.

As a comparative illustration, suppose we were given an instance of the long integer multiplication problem, encoded in words, but chose to operate only on bits. Then the size-$n$ input would have bit-length $wn$, and our cost (in word operations) would be proportional to the latter value, reflecting the fact that every word operation was wasteful by only computing with single bits. A similar phenomenon is happening here. Observe, however, that as $w$ must be bounded by the logarithm of the input size, the effect of factors of $w$ in the complexity will not be very significant.

### 8.5.1 Complexity of Sparsest Shift Computation

Algorithm 8.1 gives our algorithm to compute the sparsest shift $\alpha$ of an unknown polynomial $f \in \mathbb{Q}[x]$ given bounds $B_A$, $B_T$, $B_H$, and $B_N$ and an oracle for choosing primes. The details of this oracle are given in Section 8.4.

To choose primes, we set $\ell = 2B_A w + 1$, and $\beta_1 = 2B_H w$ and $\beta_2 = B_N(3B_T - 1)w$ (according to Corollary 8.6 and the definitions of $\beta_1, beta_2, \ell$ in the previous section). i For the sake of notational brevity, define $B_\Sigma = (B_A + B_H + B_N B_T)w$ so that $\beta_1 + \beta_2 + \ell \in O(B_\Sigma)$.

**Theorem 8.12.** *Suppose $f \in \mathbb{Q}[x]$ is an unknown polynomial given by a black box, with bounds $B_A, B_T, B_H$, and $B_N$ given as above. If $\deg f > 2B_T$, then the sparsest shift $\alpha \in \mathbb{Q}$ of $f$ can be computed deterministically using*

$$O\left(w\, B_A\, B_\Sigma^{3.78} \cdot \log^{5.78} B_\Sigma\right)$$

*word operations, plus $O(\kappa B_\Sigma^{2.89} \log^{1.89} B_\Sigma)$ word operations for the black-box evaluations.*

*Proof.* Algorithm 8.1 will always satisfy the conditions of Step 2 and terminate after $2B_A w + 1$ good primes have been produced by the oracle.

Using the oracle to choose primes, and because $\beta_1 + \beta_2 + \ell \in O(B_\Sigma)$,

$$(B_\Sigma^{1.89} \cdot \log^{1.89} B_\Sigma \cdot \operatorname{logloglog} B_\Sigma)$$

word operations are used to compute all the primes on Step 3, by Theorem 8.11. This cost does not dominate the complexity. Also, by Theorem 8.10, each chosen $p$ is bounded by $O(B_\Sigma^{1.89} \log^{1.89} B_\Sigma)$.

All black-box evaluations are performed on Step 4. There are $p$ evaluations at each iteration, and $O(B_\Sigma)$ iterations, and observe that $p < B_\Sigma^{O(1)}$, and we can safely assume this latter value is word-sized, since $B_\Sigma$ is polynomially-related to the instance size. Therefore every operation in $\mathbb{F}_p$ takes $O(1)$ word operations, a total cost of $O(\kappa B_\Sigma p)$ word operations. Using the fact that $p \in O(B_\Sigma^{1.89} \log^{1.89} B_\Sigma)$ gives the stated result.

Steps 10–14 are never executed when $\deg f > 2B_T$. Step 15 is only executed once and never dominates the complexity.

Dense polynomial interpolation over $\mathbb{F}_p$ is performed at most $O(B_\Sigma)$ times on Step 5 and $O(p)$ times at each of $O(w B_A)$ iterations through Step 7. Since $p \gg B_\Sigma$, the latter step dominates. Using asymptotically fast methods, each interpolation of $f^{(p)}(x+\gamma)$ uses $O(\mathsf{M}(p)\log p)$ field operations in $\mathbb{F}_p$, each of which again uses just a constant number of word operations. Also, from Chapter 1, recall that we can assume $\mathsf{M}(p) \in O(p \log p)$ by encoding the polynomials in $\mathbb{F}_p[x]$ into long integers. This gives a total cost over all iterations of $O(w B_A\, p^2 \log^2 p)$ (a slight abuse of notation here since the value of $p$ varies). Again, using the fact that $p \in O(B_\Sigma^{1.89} \log^{1.89} B_\Sigma)$ gives the stated result. □

To simplify the discussion somewhat, consider the case that we have only a *single* bound on the size of the output polynomial, say $B_f \geq \text{size}(f)$. By setting each of $B_T$, $B_H$, and $B_N$ equal to $B_f$, and because $w \in O(\log \text{size}(f))$, we obtain the following comprehensive result:

**Corollary 8.13.** *If an unknown polynomial $f \in \mathbb{Q}[x]$ has shifted-lacunary size bounded by $B_f$, then the sparsest shift $\alpha$ of $f$ can be computed using $O\left(B_f^{8.56} \log^{10.56} B_f\right)$ word operations, plus $O\left(\kappa B_f^{5.78} \log^{4.78} B_f\right)$ word operations for the black-box evaluations.*

*Proof.* The stated complexities follow directly from Theorem 8.12 above, using the fact that $B_\Sigma \in O(w B_f^2)$. Using the single bound $B_f$, we see that these costs always dominate the cost of Steps 10–14 given in Theorem 8.3, and so we have the stated general result. □

In fact, if we have no bounds at all *a priori*, we could start by setting $B_f$ to some small value (perhaps dependent on the size of the black box or $\kappa$), running Algorithm 8.1, then doubling $B_f$ and running the algorithm again, and so forth until the same polynomial $f$ is computed in successive iterations. This can then be tested on random evaluations. Such an approach yields an output-sensitive polynomial-time algorithm which should be correct on most input, though it could certainly be fooled into early termination.

Somewhat surprisingly, our algorithm is competitive even with the best-known sparsest shift algorithms which require a (dense) $f \in \mathbb{Q}[x]$ to be given explicitly as input. By carefully constructing the modular black box from a given $f \in \mathbb{Q}[x]$, and being sure to set $B_T < (\deg f)/2$, we can derive from Algorithm 8.1 a deterministic sparsest-shift algorithm with bit complexity close to the fastest algorithms in Giesbrecht et al. (2003); the dependence on degree $n$ and sparsity $t$ will be somewhat less, but the dependence on the size of the coefficients size$(f)$ is greater.

To understand the limits of our computational techniques (as opposed to our current understanding of the least prime in arithmetic progressions) we consider the cost of our algorithms under the assumption that $S(q) \in O(q \log^2 q)$, as discussed in the previous section. In this case the sparsest shift $\alpha$ of an unknown polynomial $f \in \mathbb{Q}[x]$, whose size in the sparsest-shifted representation is bounded by $B_f$, can be computed using $\tilde{O}(B_f^5)$ word operations. As noted in the previous section, we have verified computationally that $S(q) \le 2q \ln^2 q$ for all 64-bit primes $p = S(q)$. This would suggest the above complexity for all sparsest-shift interpolation problems that we would expect to encounter.

### 8.5.2 Complexity of Interpolation

The complexity analysis of the sparse interpolation algorithm given in Algorithm 8.2 will be quite similar to that of the sparsest shift algorithm above. Here, we need

$$\ell = w \cdot \max\{2B_H + 1, B_N\}$$

good primes to satisfy the conditions of Step 3, and from Theorem 8.8, we set $\beta_1 = 2w B_H B_T$ and $\beta_2 = \frac{1}{2} w B_N B_T (B_T - 1)$. Hence for this subsection we set

$$B_S = w B_T (B_H + B_N B_T),$$

so that $\beta_1 + \beta_2 + \ell \in O(B_S)$.

**Theorem 8.14.** *Suppose $f \in \mathbb{Q}[x]$ is an unknown polynomial given by a modular black box, with bounds $B_T$, $B_H$, $B_N$, and $B_S$ given as above. The sparse representation of $f$ as in* (8.2) *can be computed with $O(B_S^{2.89} \log^{3.89} B_S)$ word operations, plus $O(\kappa B_S^{2.89} \log^{1.89} B_S)$ word operations for the black-box evaluations.*

*Proof.* As in the sparsest-shift computation, the cost of choosing primes in Step 4 is

$$O(B_S^{1.89} \log^{1.89} B_S \cdot \operatorname{logloglog} B_S)$$

word operations, and each chosen prime $p_k$ satisfies $p_k \in O(B_S^{1.89} \log^{1.89} B_S)$.

The cost of Step 5 is $O(p \log^2 p)$ word operations at each of the $O(B_S)$ iterations, for a total cost of $O(p B_S \log^2 p)$ word operations. The black-box evaluations all occur on this step, and their total cost is $O(\kappa p B_S)$ word operations, which gives the stated cost.

We can compute each $g^{(p_i)}$ in Step 11 using $O(\mathsf{M}(t) \log t)$ ring operations modulo $p_i - 1$. Note that $k$ is bounded by $O(\ell)$, which in turn is $O(w \cdot (B_H + B_N))$. This gives the total cost in word operations for all iterations of this step as $O(w \cdot (B_H + B_N) B_T \log^2 B_T)$.

Step 12 performs $t$ Chinese Rememainderings each of $k$ modular images, and the size of each resulting integer is bounded by $B_N$, for a cost of $O(B_T B_N \log^2 B_N)$ word operations.

To factor $g$ in Step 13, we can use Algorithm 14.17 of von zur Gathen and Gerhard (2003), which has a total cost in word operations of

$$O\left(B_T^3 B_N^2 \log^2 B_T \cdot (\log B_T + \log B_N)^2\right)$$

because the degree of $g$ is $t$, $g$ has $t$ distinct roots, and each coefficient has size bounded by $O(B_T B_N)$.

In Step 16, we must first compute the modular image of $e_i \bmod p_j - 1$ and then look through all $t$ exponents of $f^{(p_j)}$ to find a match. This is repeated $tk$ times. We can use fast modular reduction to compute all the images of each $e_i$ using $O(B_N \log^2 B_N)$ bit operations, so the total cost is $O(w B_T^2(B_H + B_N) + B_T B_N \log^2 B_N)$ word operations.

Finally, we perform Chinese remaindering and rational reconstruction of $t+1$ rational numbers, each of whose size is bounded by $B_H$, for a total cost of $O(B_T B_H \log^2 B_H)$ word operations.

Therefore we see that the complexity is always dominated by the dense interpolation in Step 5. □

Once again, by having only a single bound on the size of the output, the complexity measures are greatly simplified.

**Corollary 8.15.** *If the lacunary representation size of an unknown polynomial $f \in \mathbb{Q}[x]$ is bounded by $B_f$, then that representation can be interpolated from a modular black box using $O\left(B_f^{8.67} \log^{6.78} B_f\right)$ word operations, plus $O\left(\kappa B_f^{8.67} \log^{4.78} B_f\right)$ word operations for the black box evaluations.*

Similar improvements to those discussed at the end of Section 8.5.1 can be obtained under stronger (but unproven) number theoretic assumptions.

## 8.6 Conclusions and Future Work

Here we provide the first algorithm to interpolate an unknown univariate rational polynomial into the sparsest shifted power basis in time polynomial in the size of the output. The main tool we have introduced is mapping down modulo small primes where the sparse shift is also mapped nicely. This technique could be useful for other problems involving sparse polynomials as well, although it is not clear how it would apply in finite domains where there is no notion of "size".

The complexity of our algorithm is now fairly good compared to previous approaches (that required $f$ to be given explicitly as input). However, as mentioned, improved results bounding the size of primes in arithmetic progressions could improve the provable complexity of our algorithm a bit further. A different approach suggested by the result of Baker and Harman (1996) might also be useful to us in choosing smaller primes. Rather than start with primes $q$ and find larger primes $p$ congruent to 1 modulo $q$, as we have done, they start with the larger prime $p$ and give lower bounds on the greatest prime divisor of $p-1$. It is possible that this type of result could improve our complexity, but we have not yet fully investigated the possibility.

There are many further avenues to consider, the first of which might be multivariate polynomials with a shift in each variable (see, e.g., Grigoriev and Lakshman (2000)). It would be easy to adapt our algorithms to this case provided that the degree in *each variable* is more than twice the sparsity (this is called a "very sparse" shift in Giesbrecht et al. (2003)). In such cases, we could just choose random fixed values for all variables but one, then perform the univariate sparsest shift algorithm to find the sparsest shift of that variable, and proceed to all other variables.

Finding multivariate shifts in the general case seems more difficult. The precise difficult case here is actually similar to the "Zippel sparse" complexity model, when the sparsity of the sparsest shift is proportional to the partial degrees in each variable. Since the shifts may not even be unique in such situations, it seems that a new approach will be necessary for this problem.

Even more challenging would be allowing multiple shifts, for one or more variables — for example, finding sparse $g_1, \ldots, g_k \in \mathbb{Q}[x]$ and shifts $\alpha_1, \ldots, \alpha_k \in \mathbb{Q}$ such that the unknown polynomial $f(x)$ equals $g_1(x-\alpha_1)+\cdots+g_k(x-\alpha_k)$. The most general problem of this type, which we are very far from solving, might be to compute a minimal-length formula or minimal-size algebraic circuit for an unknown function. We hope that the small step taken here might provide some insight towards this ultimate goal.

# Bibliography

Martín Avendaño, Teresa Krick, and Ariel Pacetti. Newton-Hensel interpolation lifting. *Found. Comput. Math.*, 6(1):81–120, 2006. ISSN 1615-3375.
doi: 10.1007/s10208-005-0172-3. Referenced on page 146.

Eric Bach. Number-theoretic algorithms. *Annual Review of Computer Science*, 4(1):119–172, 1990.
doi: 10.1146/annurev.cs.04.060190.001003. Referenced on page 150.

R. C. Baker and G. Harman. The Brun-Titchmarsh theorem on average. In *Analytic number theory, Vol. 1 (Allerton Park, IL, 1995)*, volume 138 of *Progr. Math.*, pages 39–103. Birkhäuser Boston, Boston, MA, 1996. Referenced on page 155.

Markus Bläser, Moritz Hardt, Richard J. Lipton, and Nisheeth K. Vishnoi. Deterministically testing sparse polynomial identities of unbounded degree. *Information Processing Letters*, 109(3):187 – 192, 2009. ISSN 0020-0190.
doi: 10.1016/j.ipl.2008.09.029. Referenced on pages 139 and 145.

A. Borodin and I. Munro. *The computational complexity of algebraic and numeric problems.* Number 1 in Elsevier Computer Science Library; Theory of Computation Series. American Elsevier Pub. Co., New York, 1975. ISBN 0444001689 0444001565. Referenced on page 141.

Allan Borodin and Prasoon Tiwari. On the decidability of sparse univariate polynomial interpolation. *Computational Complexity*, 1:67–90, 1991. ISSN 1016-3328.
doi: 10.1007/BF01200058. Referenced on page 138.

Paul Erdös, Carl Pomerance, and Eric Schmutz. Carmichael's lambda function. *Acta Arith.*, 58 (4):363–385, 1991. ISSN 0065-1036. Referenced on page 151.

Sanchit Garg and Éric Schost. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science*, 410(27-29):2659–2662, 2009. ISSN 0304-3975.
doi: 10.1016/j.tcs.2009.03.030. Referenced on page 146.

Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra.* Cambridge University Press, Cambridge, second edition, 2003. ISBN 0521826462. Referenced on page 154.

Mark Giesbrecht and Daniel Roche. Interpolation of shifted-lacunary polynomials. *Computational Complexity*, 19:333–354, 2010. ISSN 1016-3328.
doi: 10.1007/s00037-010-0294-0. Referenced on page 137.

Mark Giesbrecht and Daniel S. Roche. Interpolation of shifted-lacunary polynomials [extended abstract]. In *Proc. Mathematical Aspects of Computer and Information Sciences (MACIS) 2007*, 2007. Referenced on pages 137 and 145.

Mark Giesbrecht, Erich Kaltofen, and Wen-shin Lee. Algorithms for computing sparsest shifts of polynomials in power, chebyshev, and pochhammer bases. *Journal of Symbolic Computation*, 36(3-4):401 – 424, 2003. ISSN 0747-7171.
doi: 10.1016/S0747-7171(03)00087-7. ISSAC 2002. Referenced on pages 138, 141, 142, 143, 144, 153 and 155.

Andrew Granville and Carl Pomerance. On the least prime in certain arithmetic progressions. *Journal of the London Mathematical Society*, s2-41(2):193–200, April 1990.
doi: 10.1112/jlms/s2-41.2.193. Referenced on page 150.

Dima Grigoriev and Marek Karpinski. A zero-test and an interpolation algorithm for the shifted sparse polynomials. In Gérard Cohen, Teo Mora, and Oscar Moreno, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 673 of *Lecture Notes in Computer Science*, pages 162–169. Springer Berlin / Heidelberg, 1993.
doi: 10.1007/3-540-56686-4_41. Referenced on page 138.

Dima Yu. Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 166–172, October 1987.
doi: 10.1109/SFCS.1987.56. Referenced on page 146.

Dima Yu. Grigoriev and Y. N. Lakshman. Algorithms for computing sparse shifts for multivariate polynomials. *Applicable Algebra in Engineering, Communication and Computing*, 11:43–67, 2000. ISSN 0938-1279.
doi: 10.1007/s002000050004. Referenced on page 155.

D. R. Heath-Brown. Almost-primes in arithmetic progressions and short intervals. *Mathematical Proceedings of the Cambridge Philosophical Society*, 83(03):357–375, 1978.
doi: 10.1017/S0305004100054657. Referenced on page 150.

D. R. Heath-Brown. Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. *Proc. London Math. Soc.*, s3-64(2):265–338, March 1992.
doi: 10.1112/plms/s3-64.2.265. Referenced on page 150.

Gerhard Jaeschke. On strong pseudoprimes to several bases. *Math. Comp.*, 61(204):915–926, 1993.
doi: 10.1090/S0025-5718-1993-1192971-8. Referenced on page 151.

Erich Kaltofen. Notes on polynomial and rational function interpolation. Unpublished manuscript, 1988. Referenced on page 146.

Erich Kaltofen and Wen-shin Lee. Early termination in sparse interpolation algorithms. *Journal of Symbolic Computation*, 36(3-4):365–400, 2003. ISSN 0747-7171.
doi: 10.1016/S0747-7171(03)00088-9. ISSAC 2002. Referenced on page 138.

Erich Kaltofen, Y. N. Lakshman, and John-Michael Wiley. Modular rational sparse multivariate polynomial interpolation. In *Proceedings of the international symposium on Symbolic and algebraic computation*, ISSAC '90, pages 135–139, New York, NY, USA, 1990. ACM. ISBN 0-201-54892-5.
doi: 10.1145/96877.96912. Referenced on page 146.

Donald E. Knuth. *The art of computer programming, Volume 2: seminumerical algorithms*. Addison-Wesley, Boston, MA, 1981. ISBN 0-201-89684-2. Referenced on page 150.

Y. N. Lakshman and B. David Saunders. Sparse shifts for univariate polynomials. *Applicable Algebra in Engineering, Communication and Computing*, 7:351–364, 1996. ISSN 0938-1279.
doi: 10.1007/BF01293594. Referenced on pages 138 and 141.

Hiroshi Mikawa. On primes in arithmetic progressions. *Tsukuba Journal of Mathematics*, 25 (1):121–153, June 2001.
URL http://hdl.handle.net/2241/100471. Referenced on pages 149 and 150.

J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.
URL http://projecteuclid.org/euclid.ijm/1255631807. Referenced on page 149.

Jr. Wagstaff, Samuel S. Greatest of the least primes in arithmetic progressions having a given modulus. *Math. Comp.*, 33:1073–1080, 1979.
doi: 10.1090/S0025-5718-1979-0528061-7. Referenced on page 151.

Triantafyllos Xylouris. On Linnik's constant. Technical report, arXiv:0906.2749v1 [math.NT], 2009.
URL http://arxiv.org/abs/0906.2749. Referenced on page 150.