Supersparse Black Box Rational Function Interpolation*

Erich L. Kaltofen
Dept. of Mathematics, NCSU
Raleigh, North Carolina 27695-8205,USA
kaltofen@math.ncsu.edu www.kaltofen.us

Michael Nehring
Dept. of Mathematics, NCSU
Raleigh, North Carolina 27695-8205,USA
michaelnehring@yahoo.com

ABSTRACT

We present a method for interpolating a supersparse blackbox rational function with rational coefficients, for example, a ratio of binomials or trinomials with very high degree. We input a blackbox rational function, as well as an upper bound on the number of non-zero terms and an upper bound on the degree. The result is found by interpolating the rational function modulo a small prime p, and then applying an effective version of Dirichlet's Theorem on primes in an arithmetic progression progressively lift the result to larger primes. Eventually we reach a prime number that is larger than the inputted degree bound and we can recover the original function exactly. In a variant, the initial prime p is large, but the exponents of the terms are known modulo larger and larger factors of $p-1$.

The algorithm, as presented, is conjectured to be polylogarithmic in the degree, but exponential in the number of terms. Therefore, it is very effective for rational functions with a small number of non-zero terms, such as the ratio of binomials, but it quickly becomes ineffective for a high number of terms.

The algorithm is oblivious to whether the numerator and denominator have a common factor. The algorithm will recover the sparse form of the rational function, rather than the reduced form, which could be dense. We have experimentally tested the algorithm in the case of under 10 terms in numerator and denominator combined and observed its conjectured high efficiency.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Algorithms*

General Terms: algorithms, experimentation

Keywords: lacunary polynomials, Cauchy interpolation, sparse solution vectors

1. INTRODUCTION

A supersparse (lacunary) polynomial g over the integers $\mathbb{Z}$ in n variables with τ terms is represented as a sum (list) of monomials $g(x_1,\ldots,x_n) = \sum_{i=1}^{\tau} c_i \prod_{j=1}^{n} x_j^{e_{i,j}}$ where each monomial is represented as a coefficient $c_i \neq 0$ and the term degree vector $[\ldots, e_{i,j}, \ldots]$. The bit storage size, using "dense" binary numbers for the $e_{i,j}$, is $O(\sum_i \log(|c_i|+2) + \sum_{i,j} \log(e_{i,j}+2))$, hence proportional to the *logarithm* of the total degree $\deg(g) = \max_i \sum_j e_{i,j}$. Although several problems in polynomial algebra, such as greatest common divisor and root finding in a finite field, are NP-hard with respect to this compact size measure [26, 22], important other problems such as computing low degree factors and sparse roots have polynomial time algorithms [3, 23, 16, 4, 8, 9].

Here we consider the problem of interpolating a fraction of two supersparse polynomials $f = g/h$ over $\mathbb{Z}$. Again we seek algorithms that have polynomial running time in the supersparse size of the interpolant polynomials g and h. Since the values of a supersparse polynomial at integer points may have exponentially many bits in the size of the polynomial, we shall assume that the values can be obtained modulo any large prime p. That is, we have an algorithm ("black box") for evaluating f at *any* point modulo p. If the denominator h evaluates to 0, the black box returns ∞. This idea is illustrated below:

$$\xrightarrow{\gamma \in \mathbb{Z}_p^n,\ p} \underset{g,h \in \mathbb{Z}[x_1,\ldots,x_n]}{\blacksquare} \xrightarrow{f(\gamma) = \left(\frac{g}{h}(\gamma) \bmod p\right) \in \mathbb{Z}_p \cup \{\infty\}}$$

Note that a straight-line program representation for g/h [13] provides such a black box. We further assume that we have accurate (but not necessarily tight) bounds on the number of terms, the degree, and the coefficients.

Dense rational function interpolation goes back to Cauchy. Multivariate sparse rational function interpolation with algorithms that are polynomial in the degrees and number of variables and terms in g and h are presented in [13, 18, 21]. An important ingredient are Zippel's or Ben-Or and Tiwari's sparse multivariate *polynomial* interpolation algorithms (see, e.g., [17] and the references there). A variant of Ben-Or and Tiwari's polynomial interpolation modulo large primes p, where $p-1$ have only small prime factors and hence one has a fast discrete logarithm [27], was given by Kaltofen in 1988 (see [14, 15]). Already there we observed that by Kronecker's substitution the multivariate problem reduces to the univariate problem (see Section 7.1 below). In [5] the supersparse polynomial interpolation problem is solved by Chinese remaindering the symmetric functions of

*This research was supported in part by the National Science Foundation under Grants CCF-0830347, CCF-0514585 and DMS-0532140.

ISSAC'11, June 8–11, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0675-1/11/06 ...$10.00.

the term exponents modulo small primes without the need of special primes.

Our algorithm consists of two major steps. The first step is to recover the rational function modulo a small prime. The second step is then to lift that rational function to progressively larger primes. The first step constitutes computing a sparse null space vector modulo a small prime of a Vandermonde-like matrix. At this time, we can execute this step efficiently only if there are a few terms. Unlike Cauchy interpolation, the critical input of this algorithm is a bound on the number of terms, not a bound on the degrees. If the bound is sufficiently tight, then the algorithm will recover a sparse form of the rational function (see Section 7.2).

A surprising property is that our algorithm recovers the sparse form of the function f even if $\mathrm{GCD}(g, h)$ is non-trivial. For instance, our algorithm can reconstruct the sparse fraction representation $(x^{2^\delta} - 1)/(x - 1)$ from a black box of the polynomial $\sum_{0 \le i < 2^\delta} x^i$ in polynomial time in δ. After learning about our result, [10] considered the problem of the sparsest polynomial multiple when the denominator of the fractional representation *of a polynomial* can be dense.

Our methods are adaptable to supersparse vector rational function recovery in the sense of [25], or computing a sparsest shift for the supersparse rational function in the sense of [6, 7], but we do not discuss those generalization here. We will focus on the univariate case, as the multivariate case is handled via Kronecker substitution.

Notation: For a prime p we have

$$x^e \equiv x \cdot x^{(e-1) \bmod (p-1)} \pmod{x^p - x}, \quad \text{for} \quad e \ge 1, \quad (1)$$

which preserves the sparsity of a polynomial. For our evaluations $\gamma \in \mathbb{Z}_p$ we have $\gamma^p = \gamma$ and we shall take the liberty to omit $x^p - x$ from our polynomial congruences such as $x^p \equiv x \pmod{p}$. Note that in (1) the exponent $e \bmod (p-1)$ is represented in a residue system $\{1, \dots, p-1\}$.

Outline of approach: Suppose $f(x) = g(x)/h(x)$ is the supersparse fraction. Our algorithm chooses a relatively small random prime p and constructs a sparse fraction modulo $x^p - x$ by evaluating at $\gamma \in \mathbb{Z}_p$ and computing a sparse null space vector in the arising linear system

$$\Big(g(x) \bmod (x^p - x)\Big)(\gamma) \equiv f(\gamma)\Big(h(x) \bmod (x^p - x)\Big)(\gamma),$$

(see Section 2). We then lift the exponents from one prime p to the next prime p' by ensuring that $p' - 1$ is divisible by $p - 1$ so that the new exponent candidates in (1) are restricted to

$$1 + ((e-1) \bmod (p-1)) + i(p-1), \; 0 \le i < (p'-1)/(p-1). \quad (2)$$

(see Section 3). Such prime sequences exist due to effective versions of Dirichlet's Theorem on primes in an arithmetic progression (see Section 3.1). Once the term exponents are known, the rational coefficients are recovered by rational vector recovery. Alternatively, we may reduce the range of (2) to 2 choices by evaluating at powers of γ in the manner of the Silver-Pohlig/Hellman discrete logarithm algorithm (see Section 4).

2. COMPUTING A MODULAR IMAGE OF THE SUPERSPARSE FRACTION

First, we compute the fraction modulo a small prime. In Section 3 we enlarge that modulus.

2.1 A System of Linear Equations

Since f has a representation as a rational function, $f(x) = g(x)/h(x)$ let f be represented as follows:

$$f(x) = \Big(\sum_{i=0}^{d} a_i x^{i+1}\Big) / \Big(\sum_{i=0}^{d} b_i x^{i+1}\Big).$$

Note that the numerator and denominator are divisible by x, which does not affect the sparsity. There may be further common factors. The black box allows us to evaluate f at any point and get the value modulo p. Modulo p, $\gamma^p = \gamma$, so one only needs to concern one's self with the exponents from 1 to $p-1$ in the numerator and denominator before any wrap-around occurs. Note that there is no constant term in the numerator or denominator because we multiplied numerator and denominator by x to remove the constant term, and no other exponent maps to the constant term. This gives us the following:

$$f(x) \equiv \Big(\sum_{i=1}^{p-1} \alpha_i x^i\Big) / \Big(\sum_{i=1}^{p-1} \beta_i x^i\Big) \pmod{x^p - x, p}.$$

Clearing the denominators and subtracting the left side from the right yields the following.

$$\sum_{i=1}^{p-1} \alpha_i x^i - f(x) \times \Big(\sum_{i=1}^{p-1} \beta_i x^i\Big) \equiv 0 \pmod{x^p - x, p}$$

As in [21], for each evaluation of $x = \gamma$, $\gamma \in 1, 2, \dots, p-1$, this produces a linear equation. Indeed, even for values for which the function is undefined there is a linear equation for the denominator. The resulting system of equations has the form $A = [V \mid DW]$, where V and W are Vandermonde matrices and D is a diagonal matrix. The matrix V will have a row of zeros whenever the residue of evaluation causes the function to be undefined. The matrix D has the values of the function f along the diagonal, and a 1 where for rows that correspond to a residue γ that makes $f(\gamma)$ undefined. The coefficient list of the rational function is a vector in the null space of that matrix. Unfortunately, for any prime p there are only $p-1$ residues to work with and $2p-2$ unknowns. There are only $p-1$ useful residues, since 0 provides no useful information because we know in advance that the function will be undefined at 0. It is easy to see that the resulting matrix has full rank and therefore the null space has dimension $p-1$. We wish to find a sparse vector in that null space. Finding the sparsest vector in a linear subspace is in general NP-hard [2], and we have observed no obvious property in the resulting null space that would make it much easier to find a sparse null space vector. It should be noted however if $g(x)/h(x)$ is a rational representation of the given function, then so is $x \times g(x)/(x \times h(x))$, $x^2 \times g(x)/(x^2 \times h(x))$, and so forth. This means, assuming that no unexpected wrap-around occurs, there will be $p-1$ sparse null space vectors, where one vector is the same as the other, except that the coefficients are shifted to the right and then wrapped around appropriately. That means that the null space contains many very sparse vectors and thus one can hope that they will not be too difficult to find.

2.2 Finding a Sparse Vector

The problem of finding the sparsest null space vector for any given matrix has been shown to be NP-hard. Due to the nature of the matrix being investigated, it may be possible to create an algorithm to find sparse vectors in the null space by exploiting the structure of the matrix and null space. However, we have not yet found such an algorithm.

The simplest exhaustive search technique is pick τ many columns and guess that those columns correspond to the non-zero coefficients. Then, we look in the null space to see if there is a vector where all other entries are zero. If so, then we have found a vector with the desired sparsity. If not, then we have to pick a different set of τ columns. We do this until we have searched all possible combinations. However, in practice, this technique is not as fast the following technique.

The current strategy, which is effective for a small number of non-zero terms, is to guess new linear equations. We currently have $p-1$ equations and $2p-2$ unknowns. However, we are told that most of the coefficients are zero. A coefficient being zero represents an additional linear equation. We are not told, however, which specific coefficients are zero. Therefore, the strategy is to pick at least $p-1$ coefficients at random and set them equal to zero. One may have to pick more if any unexpected equation dependencies occur. We pick enough coefficients that the dimension of the resulting null space is precisely 1. If we only picked only correct coefficients, then the null space vector will indeed be our desired sparse null space vector. If, however, we set coefficients equal to zero that are in fact non-zero, then the resulting null space vector will be an incorrect dense vector. We repeat this process until we find a vector whose density is no higher than the inputted bound. If we exhaust all combinations, then we conclude the bound is incorrect and return FAIL.

This process is exponential in the number of non-zero terms. We can note that one can decrease the chances of setting a non-zero coefficient to zero by picking a large p, because the pool of zero coefficients will increase, but the number of non-zero coefficients remains the same, thus decreasing the chances that we mistaken try to set a non-zero coefficient to zero. However, increasing p will also increase the computational cost of the algorithm.

At this point it should be noted that this process is more likely to find the sparsest vector, as opposed to a less sparse vector. The process forces certain coefficients to be zero, and determines the remaining coefficients. If there is a very sparse form, and a somewhat less sparse form, the process is more likely to hit a non-zero coefficient of the less sparse form, because it has more non-zero coefficients to hit.

Take for example the following function, mod 11, in the following two forms:

$$\frac{x^9-1}{x-1} = \frac{x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1}{1}.$$

The first sparse form has only 4 non-zero terms, and the second model has 10 non-zero terms. So, in randomly setting coefficients to zero, the process is more likely to eliminate a coefficient corresponding to one of the 10 non-zero terms in the dense form than it is to eliminate a coefficient corresponding to one of the 4 non-zero terms in the sparse form. Therefore, this algorithm is unaware of whether the numerator and denominator have a common factor, and it will generally find the sparsest form before it finds other forms.

2.3 Uniqueness of the Sparsest Vector

One concern that this algorithm brings up is whether the sparsest vector in the null space is the vector we desire. In fact, this is not always the case. If $g(x)/h(x)$ and $\bar{g}(x)/\bar{h}(x)$ are both rational functions in the null space of the given matrix, then so is the following:

$$\Big(\lambda g(x) + \bar{\lambda}\bar{g}(x)\Big)\Big/\Big(\lambda h(x) + \bar{\lambda}\bar{h}(x)\Big), \quad \lambda, \bar{\lambda} \in \mathbb{Z}.$$

So, if there are two rational functions in the null space that have the same monomials as support, then it would be possible to eliminate one of those monomials by taking a linear combination of the two rational functions, thus resulting in a sparser function. Every rational function has a sparsest representation over the rational numbers, and that sparsest representation corresponds to a sparse representation modulo any given prime. If there is another vector in the null space with the same support, then a linear combination would result in a sparser vector for that prime. Also, for a finite number of primes, there is also the possibility that two non-zero monomials will collide, either to a single monomial, which is easy to deal with, or they may destructively collide to zero, which we have to deal with probabilistically later.

It is possible to construct examples where a given rational function has a sparser representation modulo a given prime. Here is one example to illustrate what happens. Take the rational function $(x^{34}+4x)/(2x-3x^{40})$ If we multiply the numerator and denominator by x^3, we get the same rational function in another form, $(x^{37}+4x^4)/(2x^4-3x^{43})$. Now, we can look at both those rational functions modulo 7. Recall that modulo 7, the exponents wrap around modulo 6, so the functions are the following two functions, respectively: $(x^4+4x)/(2x-3x^4)$ and $(4x^4+x)/(2x^4-3x)$. We can now take the same linear combination of the numerator and denominator and get another equivalent rational function

$$\frac{(x^4+4x)-4(4x^4+x)}{(2x-3x^4)-4(2x^4-3x)} = \frac{-15x^4}{14x-11x^4} \equiv \frac{-x^4}{3x^4} \bmod 7.$$

So, for certain rational functions, the sparsest representation over the integers may not be the sparsest modulo a certain prime. While we do not currently have a proof of this, it seems unlikely that there are many, if any, rational functions that have a sparser representation modulo many prime numbers. In any case, the valid sparse solution will be among the sparse candidate vectors.

2.4 An Example

Let $f(x)$ be a blackbox of the function

$$f(x) = (8x^{882704} - 3x^{6098})/(5x^{1048576} + 1).$$

We want there to be no constant terms, so $f(x)$ is the same as the following rational function, except at the point $x = 0$,

$$(8x^{882705} - 3x^{6099})/(5x^{1048577} + x).$$

We choose the prime $p = 13$, so all exponent wrap-around will occur modulo 12. We now evaluate this box black at all the residues modulo 13, except for 0.

x	1	2	3	4	5	6	7	8	9	10	11	12
$f(x)$	3	7	2	4	4	1	1	4	4	2	7	3

This data allows us to set up a system of linear equations, which is represented by the null space of the following matrix. The matrix

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\
2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 & 7 & 1 & 12 & 11 & 9 & 5 & 10 & 7 & 1 & 2 & 4 & 8 & 3 & 6 \\
3 & 9 & 1 & 3 & 9 & 1 & 3 & 9 & 1 & 3 & 9 & 1 & 7 & 8 & 11 & 7 & 8 & 11 & 7 & 8 & 11 & 7 & 8 & 11 \\
4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 & 10 & 1 & 10 & 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 \\
5 & 12 & 8 & 1 & 5 & 12 & 8 & 1 & 5 & 12 & 8 & 1 & 6 & 4 & 7 & 9 & 6 & 4 & 7 & 9 & 6 & 4 & 7 & 9 \\
6 & 10 & 8 & 9 & 2 & 12 & 7 & 3 & 5 & 4 & 11 & 1 & 7 & 3 & 5 & 4 & 11 & 1 & 6 & 10 & 8 & 9 & 2 & 12 \\
7 & 10 & 5 & 9 & 11 & 12 & 6 & 3 & 8 & 4 & 2 & 1 & 6 & 3 & 8 & 4 & 2 & 1 & 7 & 10 & 5 & 9 & 11 & 12 \\
8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 8 & 12 & 5 & 1 & 7 & 4 & 6 & 9 & 7 & 4 & 6 & 9 & 7 & 4 & 6 & 9 \\
9 & 3 & 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 3 & 1 & 3 & 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 & 3 & 1 & 9 \\
10 & 9 & 12 & 3 & 4 & 1 & 10 & 9 & 12 & 3 & 4 & 1 & 6 & 8 & 2 & 7 & 5 & 11 & 6 & 8 & 2 & 7 & 5 & 11 \\
11 & 4 & 5 & 3 & 7 & 12 & 2 & 9 & 8 & 10 & 6 & 1 & 1 & 11 & 4 & 5 & 3 & 7 & 12 & 2 & 9 & 8 & 10 & 6 \\
12 & 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 1 & 12 & 1 & 3 & 10 & 3 & 10 & 3 & 10 & 3 & 10 & 3 & 10 & 3 & 10
\end{bmatrix}$$

has the form $[VDW]$, where V and W are Vandermonde and D is a diagonal matrix whose values are the negative of the function values.

We wish to find a sparse vector in the null space of that matrix. The matrix has currently column dimension 24 and a 12 dimensional null space. The current strategy is to randomly add equations where we set one coefficient equal to zero. Since the rational function is sparse, we know that we have a reasonable chance of only adding correct equations. For visual illustrative purposes, rather than adding new rows to the matrix, we set the values in the corresponding columns to zero in

$$\begin{bmatrix}
\mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} & 1 & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 10 & \mathbf{0} & 10 & \mathbf{0} & 10 & 10 & 10 & \mathbf{0} & \mathbf{0} & 10 \\
\mathbf{0} & 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 11 & \mathbf{0} & 5 & 10 & 7 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 9 & \mathbf{0} & 10 & \mathbf{0} & 1 & 2 & 4 & \mathbf{0} & \mathbf{0} & 6 \\
\mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 3 & \mathbf{0} & 1 & 3 & 9 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 11 & \mathbf{0} & 8 & \mathbf{0} & 7 & 8 & 11 & \mathbf{0} & \mathbf{0} & 11 \\
\mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 4 & \mathbf{0} & 12 & 9 & 10 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 4 & \mathbf{0} & 12 & \mathbf{0} & 10 & 1 & 4 & \mathbf{0} & \mathbf{0} & 9 \\
\mathbf{0} & 12 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 8 & \mathbf{0} & 5 & 12 & 8 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 7 & \mathbf{0} & 6 & \mathbf{0} & 7 & 9 & 6 & \mathbf{0} & \mathbf{0} & 9 \\
\mathbf{0} & 10 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 7 & \mathbf{0} & 5 & 4 & 11 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 5 & \mathbf{0} & 11 & \mathbf{0} & 6 & 10 & 8 & \mathbf{0} & \mathbf{0} & 12 \\
\mathbf{0} & 10 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 6 & \mathbf{0} & 8 & 4 & 2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 8 & \mathbf{0} & 2 & \mathbf{0} & 7 & 10 & 5 & \mathbf{0} & \mathbf{0} & 12 \\
\mathbf{0} & 12 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 5 & \mathbf{0} & 8 & 12 & 5 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 6 & \mathbf{0} & 7 & \mathbf{0} & 6 & 9 & 7 & \mathbf{0} & \mathbf{0} & 9 \\
\mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 9 & \mathbf{0} & 1 & 9 & 3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 9 & \mathbf{0} & 1 & \mathbf{0} & 3 & 1 & 9 & \mathbf{0} & \mathbf{0} & 9 \\
\mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 10 & \mathbf{0} & 12 & 3 & 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 2 & \mathbf{0} & 5 & \mathbf{0} & 6 & 8 & 2 & \mathbf{0} & \mathbf{0} & 11 \\
\mathbf{0} & 4 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 12 & 2 & \mathbf{0} & 8 & 10 & 6 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 4 & \mathbf{0} & 3 & \mathbf{0} & 12 & 2 & 9 & \mathbf{0} & \mathbf{0} & 6 \\
\mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 12 & \mathbf{0} & 12 & 1 & 12 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 3 & \mathbf{0} & 3 & \mathbf{0} & 3 & 10 & 3 & \mathbf{0} & \mathbf{0} & 10
\end{bmatrix}$$

Now the null space of that matrix, ignoring the trivial null space vectors is the following:

$$[\mathbf{0}\ 0\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}\ 0\ 5\ \mathbf{0}\ 6\ 0\ 2\ \mathbf{0}\ \ \mathbf{0}\ \mathbf{0}\ 7\ \mathbf{0}\ 7\ \mathbf{0}\ 11\ 0\ 1\ \mathbf{0}\ \mathbf{0}\ 0]^T.$$

The bold entries correspond to the entries that we forced to be zero with our random equations. The vector has seven non-zero entries, instead of the required four non-zero entries. So we must try again with different columns, illustrated in the matrix below. Again, we look at the non-trivial null space vector of the matrix. Now it is the following:

$$[\mathbf{0}\ \mathbf{0}\ 0\ \mathbf{0}\ 0\ 8\ \mathbf{0}\ 0\ \mathbf{0}\ \mathbf{0}\ 0\ 10\ \ \mathbf{0}\ 5\ \mathbf{0}\ \mathbf{0}\ 0\ \mathbf{0}\ 0\ \mathbf{0}\ 0\ 1\ \mathbf{0}\ \mathbf{0}]^T.$$

There are four non-zero entries in the new null space. That is the desired sparsity, so we selected correct columns and found a sparse vector. This vector corresponds to the rational function $\hat{f}(x) = (8x^6 + 10x^{12})\ /\ (5x^2 + x^{10})$. The following relationship holds, where the exponents on the right are mapped according to $x^{13} \equiv x$, and the coefficients are taken modulo 13: $\hat{f}(x) \equiv f(x) \times x^{10}/x^{10} \bmod (x^13 - x, 13)$.

$$\begin{bmatrix}
\mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 1 & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 1 & 1 & \mathbf{0} & 10 & \mathbf{0} & \mathbf{0} & 10 & \mathbf{0} & 10 & \mathbf{0} & 10 & 10 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 8 & \mathbf{0} & 6 & 12 & \mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & 7 & 1 & \mathbf{0} & 11 & \mathbf{0} & \mathbf{0} & 10 & \mathbf{0} & 1 & \mathbf{0} & 4 & 8 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 9 & 1 & \mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & 9 & 1 & \mathbf{0} & 8 & \mathbf{0} & \mathbf{0} & 8 & \mathbf{0} & 7 & \mathbf{0} & 11 & 7 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 12 & \mathbf{0} & 10 & 1 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 10 & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 12 & \mathbf{0} & 10 & \mathbf{0} & 4 & 3 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 8 & \mathbf{0} & 5 & 12 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 8 & 1 & \mathbf{0} & 4 & \mathbf{0} & \mathbf{0} & 6 & \mathbf{0} & 7 & \mathbf{0} & 6 & 4 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 8 & \mathbf{0} & 2 & 12 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 11 & 1 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 11 & \mathbf{0} & 6 & \mathbf{0} & 8 & 9 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 5 & \mathbf{0} & 11 & 12 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 2 & 1 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 2 & \mathbf{0} & 7 & \mathbf{0} & 5 & 9 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 5 & \mathbf{0} & 8 & 12 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 5 & 1 & \mathbf{0} & 4 & \mathbf{0} & \mathbf{0} & 7 & \mathbf{0} & 6 & \mathbf{0} & 7 & 4 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 3 & 1 & \mathbf{0} & 3 & \mathbf{0} & \mathbf{0} & 3 & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 1 & \mathbf{0} & 3 & \mathbf{0} & 9 & 3 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 12 & \mathbf{0} & 4 & 1 & \mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & 4 & 1 & \mathbf{0} & 8 & \mathbf{0} & \mathbf{0} & 5 & \mathbf{0} & 6 & \mathbf{0} & 2 & 7 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 5 & \mathbf{0} & 7 & 12 & \mathbf{0} & 9 & \mathbf{0} & \mathbf{0} & 6 & 1 & \mathbf{0} & 11 & \mathbf{0} & \mathbf{0} & 3 & \mathbf{0} & 12 & \mathbf{0} & 9 & 8 & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & 12 & \mathbf{0} & 12 & 1 & \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & 12 & 1 & \mathbf{0} & 10 & \mathbf{0} & \mathbf{0} & 3 & \mathbf{0} & 3 & \mathbf{0} & 3 & 10 & \mathbf{0} & \mathbf{0}
\end{bmatrix}$$

3. LIFTING THE MODULAR IMAGE

3.1 Dirichlet's Theorem

Dirichlet's Theorem on Arithmetic Progressions states that for relatively prime numbers a and n, there are infinitely many prime numbers in the Arithmetic Progression $a + \lambda n$, $\lambda \in \mathbb{Z}_{\geq 0}$. In the lifting step that follows, we will need a sequence of primes of the form p_1, p_2, $p_3, \ldots, p_k$, where $p_i - 1$ divides $p_{i+1} - 1$. That is, $p_{i+1} = r_i(p_i - 1) + 1$. Such a prime is guaranteed to exist by Dirichlet's theorem, since $p_i - 1$ and 1 have no common factors. Since this particular sequence of primes will prove to be inconsequential in the final version of the algorithm, we will not discuss the distribution of such primes (see Section 4.1 instead). However, we will present a quick example, starting with 13, the prime we used in our example rational function.

i	1	2	3	4	5	6	7
p_i	13	37	73	433	1297	2593	10369
p_{i+1}	37	73	433	1297	2593	10369	72577
r_i	3	2	6	3	2	4	7

Digression: Our sequence construction above leads to the following, possibly new, number theoretic function: define $\kappa_p(n)$ as the minimum over all prime sequences starting at p and ending above n of the maximum of all multipliers in each sequence. For instance, we have the following minimal multipliers.

n	2^{10}	2^{20}	2^{40}	2^{80}	2^{160}	2^{320}	2^{640}	2^{1280}
$\kappa_2(n)$	4	5	15	16	49	53	132	224

3.2 Lifting to Larger Primes – The Easy Way

Assume that we know the rational function modulo a small prime p_1. We construct a sequence of primes $\{p_1, p_2, p_3, \ldots, p_k\}$ such that $(p_i - 1)$ divides $(p_{i+1} - 1)$ for $1 \leq i \leq k - 1$ and p_k is larger than the upper bound on the degree. Furthermore, define $r_i = (p_{i+1} - 1)/(p_i - 1)$. Such a sequence can be constructed according to Dirichlet's Theorem.

As above, we remember that $x^p \equiv x \bmod x^p - x$. Specifically, if we know the rational function mod p_i, we know the exponents of the terms modulo $p_i - 1$. Suppose, for example, that one exponent is congruent to m mod $(p_i - 1)$. Since $p_i - 1$ divides $p_{i+1} - 1$, that is $p_{i+1} - 1 = r_i(p_i - 1)$, we know that there are precisely r_i possibilities for each exponent modulo $p_{i+1} - 1$. In our case, those possibilities are

$$m, m + (p_i - 1), m + 2(p_i - 1), \ldots, m + (r_i - 1)(p_i - 1). \quad (3)$$

This means if there are τ_i known exponents modulo $x^{p_i} - x$, then there are $\tau_i \cdot r_i$ unknown exponents modulo $x^{p_{i+1}} - x$. At this point we know that the coefficients of all other terms must be zero. Therefore, as above, we can construct a system of linear equations, but this time we have only $\tau_i \cdot r_i$ many unknowns, but up to p_{i+1} equations. Let $\{\alpha_1, \ldots, \alpha_{d_1}\}$ be the computed list of possible numerator exponents modulo $x^{p_{i+1}} - x$ and $\{\beta_1, \ldots, \beta_{d_2}\}$ be the list of computed exponents of possible denominator exponents modulo $x^{p_{i+1}} - x$. Then, the following equation holds:

$$\begin{aligned} & a_1 x^{\alpha_1} + a_2 x^{\alpha_2} + \cdots + a_{d_1} x^{\alpha_{d_1}} \\ & \quad - f(x)(b_1 x^{\beta_1} + b_2 x^{\beta_2} + \cdots + b_{d_2} x^{\beta_{d_2}}) \equiv 0 \pmod{p_{i+1}}. \end{aligned}$$

For every value of $\gamma \in \mathbb{Z}_{p_{i+1}}$ this yields of a linear equation with unknowns $\{a_1, \ldots, a_{d_1}, b_1, \ldots, b_{d_2}\}$. It should be noted at this point that not all p_{i+1} equations are needed to find the null space vector.

We can bound the number of new equations by proceeding as in [21, Section 4.1]. Assume that $g(x)/h(x)$ and $\overline{g}(x)/\overline{h}(x)$ are two different rational functions whose coefficient vectors lie in the null space of the matrix. Then $g(x)\overline{h}(x) - \overline{g}(x)h(x)$ evaluates to zero for every residue that we used in building the null space. That is to say that vector representing $g(x)\overline{h}(x) - \overline{g}(x)h(x)$ is in the null space of a Vandermonde-like matrix, with the difference being that we only consider a limited number of powers of x, specifically, those that could occur in $g(x)\overline{h}(x)$, of which there are at most $d_1 d_2$ many. Therefore, it is possible that picking two separate residues will not generate a unique row in the Vandermonde-like matrix. For example, if the only powers we are considering are x^2 and x^{10} modulo 13, then the residues 4 and 9 would both generate the row $[3, 9]$. However, this Vandermonde-like matrix is a submatrix of the full Vandermonde matrix, which is non-singular, and thus by adding sufficient rows (corresponding to residues), we can also insure that our

Vandermonde-like matrix is non-singular. If γ is a primitive root of unity modulo p_{i+1}, then picking the residues γ, γ^2, $\gamma^3, \ldots$ will ensure that no row is repeated until all possible rows have been exhausted. Once we have $d_1 d_2$ many unique rows, then the Vandermonde-like matrix will have as many rows as columns, and will be non-singular. Since $g(x)\overline{h}(x) - \overline{g}(x)h(x)$ will lie in the null space of a non-singular matrix, that indicates that $g(x)\overline{h}(x) - \overline{g}(x)h(x) = 0$, and therefore $g(x)/h(x) = \overline{g}(x)/\overline{h}(x)$ up to a common factor in the numerator and denominator. As we see in the following example, having a common factor in the numerator and denominator is not bad, and in fact is expected and can be dealt with.

3.3 An Example For the Lifting Step

We will return to our previous example. Recall, we have a blackbox for the function

$$f(x) = (8x^{882704} - 3x^{6098})/(5x^{1048576} + 1).$$

In the initial step, we recovered this function as

$$\hat{f}(x) = (8x^6 + 10x^{12})/(5x^2 + x^{10}).$$

And we noted the following equality, where the exponents on the right are mapped according to $x^{13} = x$, and the coefficients are taken mod 13, $\hat{f}(x) \equiv f(x) \times x^{10}/x^{10} \bmod 13$. Note that $13-1$ divides $37-1$ and that $(37-1)/(13-1) = 3$. The exponent wrap-around occurs modulo 12. In our desired representation modulo 37, the exponent wrap-around will be modulo 36. The numerator exponents recovered from the first step are 6 and 12, and the denominator exponents are 2 and 10.
Numerator exponent pool: $\{6, 6+12, 6+24, 12, 12+12, 12+24\} = \{6, 18, 30, 12, 24, 36\}$.
Denominator exponent pool: $\{2, 2+12, 2+24, 10, 10+12, 10+24\} = \{2, 14, 26, 10, 22, 34\}$.
The remaining $36 - 6 = 30$ coefficients corresponding to other powers are known to be zero. Therefore, we essentially already have 60 linear equations, since we know that 60 coefficients are zero. Furthermore, we can garner 36 additional linear equations by evaluating the black box. Since there are only 72 unknowns, we will be able to determine the desired null space vector. For simplicity we will set up a system with only 12 unknowns corresponding to the 12 possible non-zero coefficients listed above. We now sample our blackbox to get some linear equations, as represented by the matrix below:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 30 & 30 & 30 & 30 & 30 & 30 \\ 27 & 26 & 36 & 10 & 11 & 1 & 1 & 34 & 26 & 33 & 10 & 7 \\ 26 & 10 & 1 & 26 & 10 & 1 & 31 & 2 & 14 & 20 & 29 & 15 \\ 26 & 10 & 1 & 26 & 10 & 1 & 10 & 16 & 26 & 12 & 1 & 9 \\ 11 & 10 & 36 & 26 & 27 & 1 & 8 & 17 & 6 & 22 & 23 & 35 \\ 36 & 1 & 36 & 1 & 36 & 1 & 8 & 8 & 8 & 8 & 8 & 8 \\ 26 & 10 & 1 & 26 & 10 & 1 & 28 & 4 & 21 & 3 & 25 & 30 \\ 36 & 1 & 36 & 1 & 36 & 1 & 18 & 32 & 18 & 32 & 18 & 32 \\ 10 & 26 & 1 & 10 & 26 & 1 & 35 & 8 & 22 & 23 & 17 & 6 \\ 1 & 1 & 1 & 1 & 1 & 1 & 7 & 34 & 7 & 34 & 7 & 34 \\ 1 & 1 & 1 & 1 & 1 & 1 & 12 & 9 & 12 & 9 & 12 & 9 \\ 10 & 26 & 1 & 10 & 26 & 1 & 12 & 1 & 16 & 26 & 9 & 10 \\ x^6 & x^{12} & x^{18} & x^{24} & x^{30} & x^{36} & x^2 & x^{10} & x^{14} & x^{22} & x^{26} & x^{34} \end{bmatrix}$$

(The final row labels the columns.)

We now look at the null space of that matrix:

$$[0\ \ 29\ \ 9\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 15]^T,$$
$$[9\ \ 0\ \ 0\ \ 0\ \ 0\ \ 29\ \ 0\ \ 0\ \ 0\ \ 15\ \ 1\ \ 0]^T,$$
$$[0\ \ 0\ \ 0\ \ 29\ \ 9\ \ 0\ \ 0\ \ 15\ \ 1\ \ 0\ \ 0\ \ 0]^T.$$

There are 3 vectors, and each vector is the same as the other, up to wrap-around. This is expected, since the vectors correspond to the following rational functions modulo 37 (with exponents modulo 36).

$$\frac{29x^{12} + 9x^{18}}{x^2 + 15x^{34}}, \frac{9x^6 + 29x^{36}}{15x^{22} + x^{26}}, \frac{29x^{24} + 9x^{30}}{15x^{10} + x^{14}}$$

Note that each rational function is equal to each other rational function up to being multiplied by $\frac{x^{12}}{x^{12}}$ or $\frac{x^{24}}{x^{24}}$, with exponents wrapped around modulo 36. Any of those 3 vectors can be lifted.

3.4 Controlling Wrap-Around

Up to this point we have taken the direct approach to interpolating rational functions modulo a prime. However, this gives us little control of how the exponent wrap-around occurs. Indeed, it must always occur modulo one less than a prime. This can make complexity analysis on the lifting step difficult, because it is uncertain where the next prime will land. Also, it leaves us more vulnerable to destructive wrap-around. For example, the rational function $(x^N - x)/(x-1)$ evaluates to zero at every point modulo the first 50 primes. Here $N = 1 + \mathrm{lcm}(2-1, 3-1, 5-1, 7-1, 11-1, \ldots, 223-1, 227-1, 229-1)$. Therefore, since each $p-1$ divides N for the first 50 primes, x^N will map to x for the first 50 primes and cancel with the $-x$ term, giving us the false impression that the rational function is zero. The chance of such destructive wrap-around is lowered if we do not limit ourselves to wrap-around modulo a prime minus one. Indeed, N is a 27 decimal digit number and has destructive wrap-around for all wrap-around modulo $p-1$ for $p-1 \leq 228$. If we could make our wrap-around occur modulo any n, where n is an integer, the same N would have to be 97 digits long to have destructive wrap-around modulo all $n \leq 228$.

Suppose we wish the wrap-around to occur modulo n, where n is any integer. The first step is to compute a prime of the form $p = kn + 1$, which always exists due to Dirichlet's Theorem. Now, instead of interpolating $f(x) \bmod p$, we interpolate $f(x^k) \bmod p$. Since k divides $p-1$, $f(x^k) \bmod x^p - x$ will map every exponent to a multiple of k, and have wrap-around modulo kn. We recover $f(x^k)$ as outlined earlier and divide each exponent by k and have the original function, but with the wrap-around occurring modulo n.

This can be briefly illustrated using our previous example function, $(8x^{882705} - 3x^{6099})/(5x^{1048577} + x)$. Suppose we want to know the exponents of rational function modulo 10. We first find a prime of the form $p = 10k+1$. One such prime is $31 = 3 \cdot 10 + 1$. Now, we know the following must be true: $f(x^3) = (8x^{3\times 882705} - 3x^{3\times 6099})/(5x^{3\times 1048577} + x^{3\times 1})$. We have a black box of $f(x^3)$. We simply evaluate at the cube of each residue. Thus, we see that $f(x^3)$ is equivalent to the following, modulo 31: $f(x^3) \equiv (8x^{15} + 28x^{27})/(5x^{21} + x^3)$. Note that the coefficients are taken modulo 31 and the exponent wrap-around occurs modulo 30. We divide each of the exponents by 3 to get back $f(x)$: $f(x) \equiv (8x^5 + 28x^9)/(5x^7 + x)$. And indeed we see that the exponents match modulo 10. This technique can be used both in the initial step, as illustrated by example here, or in the lifting step, as we will illustrate next.

4. SILVER-POHLIG/HELLMAN LIFTING

We now discuss how to reduce the number of candidate exponents in each lifting step (3) by adapting the powering technique of the Pohlig and Hellman [27] discrete logarithm algorithm, whose independent discovery they also attribute to Roland Silver (and Richard Schroeppel and H. Block).

4.1 Dirichlet's Theorem Revisited

Dirichlet's Theorem on Arithmetic Progressions states that for relatively prime numbers a and n, there are infinitely many prime numbers in the Arithmetic Progression $a + \lambda n$, $\lambda = 1, 2, \ldots$ In our algorithm we are interested in prime numbers p_j such that $2^j(p_0 - 1)$ divides $p_j - 1$, where p_0 is the initial prime that we picked. That is, we want primes of the form $p_j = m_j\, 2^j(p_0 - 1) + 1$. Since 1 and $n = 2^j(p_0 - 1)$ are relatively prime, Dirichlet's Theorem guarantees the existence of such prime numbers. Furthermore, we know there is a polynomial bound on the first such prime. Specifically, $p_j = O(n^L)$ for some Linnik's constant L $(= 5.5)$ (smaller bounds [24] exclude a zero-density set of multipliers n) or, assuming the Generalized Riemann Hypothesis holds, $p_j = O(\varphi(n)^2(\ln n)^2)$ where φ is Euler's totient function [12]. Heath-Brown [11] had stated earlier that the conjecture $L = 2$ "may presumably be reduced to $\ll n(\log n)^2$." Note that the later conjecture, as well as similar others, are asymptotic and do not provide effective estimates for the big-O constant implied by $\ll$. Since primality testing is computationally inexpensive, and because *in practice* the first prime generally occurs early in the sequence, finding such sequences of primes is feasible.

For $p_0 - 1 = 12$ we can have the following sequence:

j	0	1	2	3	4	5	6	7
$p_j - 1$ is div. by	12	24	48	96	192	384	768	1536
p_j	13	73	97	97	193	769	769	7681
m_j	1	3	2	1	1	2	1	5

The reader may have also already observed that one can pick a single prime number, the last prime in the sequence, and simply change the multipliers, as shown below:

j	0	1	2	3	4	5	6	7
$p_j - 1$ is div. by	12	24	48	96	192	384	768	1536
p_j	7681	7681	7681	7681	7681	7681	7681	7681
m_j	640	320	160	80	40	20	10	5

The latter is useful when a large number of evaluation points γ or a large modulus are needed at the start. In the single prime case, p_0 need not be a prime number.

4.2 SPH-Lifting to Larger Primes

Assume that we know the rational function modulo a small prime p_0. We construct a sequence of primes $\{p_0, p_1, p_2, \ldots, p_k\}$ such that $2^j(p_0 - 1)$ divides $p_j - 1$ for $1 \le j \le k$ and $2^k(p_0 - 1)$ is larger than the upper bound on the degree. Such a sequence can be constructed and will likely not have excessively large primes (see the above Section 4.1). We write $p_j = m_j 2^j(p_0 - 1) + 1$. Note $m_0 = 1$.

In the first step of the algorithm, we find the supersparse representation of the function modulo $x^{p_0} - x, p_0$. This means we know the exponents modulo $p_0 - 1$ since x^{p_0} evaluates the same as x modulo p_0. In this section, we will compute the exponents modulo $2(p_0 - 1)$, $2^2(p_0 - 1), \ldots$, $2^k(p_0 - 1)$. Since $2^k(p_0 - 1)$ is then larger than degree bound, we will actually know the exponents.

Suppose, at this point, that we know the exponents of the non-zero terms modulo $2^{j-1}(p_0 - 1)$ where j is between 1 and k. The case $j = 1$ is the initial step of the algorithm. We now wish to determine the exponents of the non-zero terms modulo $2^j(p_0 - 1)$.

Recall that f was represented as $f(x) = \sum_{\eta=1}^{d+1} a_\eta x^\eta \ / \ \sum_{\eta=1}^{d+1} b_\eta x^\eta$. Now, consider evaluating f at x^{m_j}. Then we get $f(x^{m_j}) = \sum_{\eta=1}^{d+1} a_\eta x^{m_j \eta} \ / \ \sum_{\eta=1}^{d+1} b_\eta x^{m_j \eta}$. Let $\eta_{j-1} = \eta \bmod (2^{j-1}(p_0 - 1))$ in the residue system $\{1, ..., (p_{j-1} - 1)/m_{j-1}\}$, where $(p_{j-1} - 1)/m_{j-1} = 2^{j-1}(p_0 - 1)$, and let $\zeta_j = (m_j \eta) \bmod (p_j - 1)$ in the residue system $\{1, \ldots, p_j - 1\}$ (see (1)). Since m_j divides $p_j - 1$, all ζ_j are divisible by m_j. Furthermore $\eta_j = \zeta_j / m_j = \eta \bmod (2^j(p_0 - 1))$, where $(p_j - 1)/m_j = 2^j(p_0 - 1)$, and either $\eta_j = \eta_{j-1}$ or $\eta_j = \eta_{j-1} + 2^{j-1}(p_0 - 1)$.

Now, we wish to exploit our knowledge of the term exponents modulo $2^{j-1}(p_0 - 1)$ to help us compute the exponents modulo $2^j(p_0 - 1)$. If x^e, $e \in \{1, 2, \ldots, 2^{j-1}(p_0 - 1) - 1\}$ has coefficient zero modulo $2^{j-1}(p_0 - 1)$, then we know both x^e and $x^{e + 2^{j-1}(p_0 - 1)}$ have coefficient zero modulo $2^j(p_0 - 1)$. We are temporally ignoring the case where those two terms have coefficients that sum to zero. Similarly, suppose we have a complete set of residues of the exponents for the non-zero terms modulo $2^{j-1}(p_0 - 1)$, say $\{e_1, e_2, \ldots, e_\tau\}$, then we know the only possible terms with non-zero coefficients will have exponents $\{e_1, e_2, \ldots, e_\tau\} \cup \{e_1 + 2^{j-1}(p_0 - 1), e_2 + 2^{j-1}(p_0 - 1), \ldots, e_\tau + 2^{j-1}(p_0 - 1)\}$ modulo $2^j(p_0 - 1)$.

When we set up our new linear equation modulo p_j, we know in advance that most of the coefficients are going to be zero. We have at most 2τ possible terms that are going to be non-zero, but up to $(p_j - 1)/m_j = 2^j(p_0 - 1)$ equations. Therefore, the number of equations will far exceed the number of unknowns, and our system will likely have fullest possible rank.

With all that said, we can now formalize what we said into a system of linear equations. Let $\{e_1^{[num]}, e_2^{[num]}, \ldots, e_{\tau^{[num]}}^{[num]}\}$ be the exponents of the non-zero terms of the numerator modulo $2^{j-1}(p_0 - 1)$. Similarly, let $\{e_1^{[den]}, e_2^{[den]}, \ldots, e_{\tau^{[den]}}^{[den]}\}$ be the denominator. That simply says the following.

$$f(x^{m_{j-1}}) \equiv \frac{\sum_{i=1}^{\tau^{[num]}} \alpha_i x^{m_{j-1} e_i^{[num]}}}{\sum_{i=1}^{\tau^{[den]}} \beta_i x^{m_{j-1} e_i^{[den]}}} \bmod (x^{p_{j-1}} - x, p_{j-1}).$$

Lifting the prime to p_j and the exponents to $2^j(p_0 - 1)$ yields the following $\bmod(x^{p_j} - x, p_j)$:

$$f(x^{m_j}) \equiv \frac{\sum_{i=1}^{\tau^{[num]}} \hat{\alpha}_i x^{m_j e_i^{[num]}} + \tilde{\alpha}_i x^{m_j (e_i^{[num]} + 2^{j-1}(p_0 - 1))}}{\sum_{i=1}^{\tau^{[den]}} \hat{\beta}_i x^{m_j e_i^{[den]}} + \tilde{\beta}_i x^{m_j (e_i^{[den]} + 2^{j-1}(p_0 - 1))}} \quad (4)$$

We also note $\hat{\alpha}_i + \tilde{\alpha}_i = \alpha_i$ and $\hat{\beta}_i + \tilde{\beta}_i = \beta_i$ if the α_i's and β_i's are known as integers or if we use a single larger modulus (see Section 6). Clearing the denominators and evaluating at various residues will create linear equations in $\hat{\alpha}_i, \tilde{\alpha}_i, \hat{\beta}_i$, and $\tilde{\beta}_i$. Note that multiplying both numerator and denominator with $x^{m_j 2^{j-1}(p_0 - 1)} = x^{(p_j - 1)/2}$ gives a second, possibly linearly independent solution coefficient vector to (4).

We will now briefly address that possibility of destructive wrap-around. For example, consider the polynomial $x^{80} - 3x^4 - x^2$. If this polynomial is looked at modulo 7, we observe that the exponent wrap-around occurs modulo 6, so the polynomial evaluates like $x^{80 \bmod 6} - 3x^{4 \bmod 6} - x^{2 \bmod 6} = x^2 - 3x^4 - x^2 = -3x^4$. So, it is possible for destructive wrap-around to occur, and that would make our rational function appear even sparser than it is. Furthermore, when we attempt to lift, we will fail, because we assume that the coefficients corresponding to x^{80} and x^2 are both zero, because the two coefficients canceled each other out modulo 6. The bad news is that we know of no elegant solution to fix this. The good news is that this can happen for only a small finite number of primes, so if destructive wrap-around causes the lifting step to fail, we can give up and try with a different initial prime and it will not take long until we have exhausted all primes for which destructive wrap-around occurs.

5. THE ALGORITHM

The reader may note at this point that we likely will never, by lifting alone, recover the actual original function, but rather an alternative representation thereof. That is because the lifting step cannot tell whether the numerator and denominator were both multiplied by a power of x. Also, the coefficients are taken modulo a prime, and may be multiplied by a constant. Furthermore, nothing will cause the exponents to match up exactly, but rather modulo the final wrap-around. However, once we have reached the degree bound and know we can stop lifting. The coefficients can easily be recovered using rational vector reconstruction [20, Section 4]. If we lift to a wrap-around that is at least twice the degree bound, which is computationally inexpensive since it is only one further lifting step, then we can easily compute an exponent shift such that all the exponents after shifting fall below the degree bound. If there are multiple such shifts, then one can sample the trial function and the blackbox at random points and compare. The function that corresponds to the blackbox is correct.

Outline of Algorithm

```
Input   A blackbox of a supersparse rational function $f \in \mathbb{Q}(x)$, an upper bound on the number of non-zero coefficients $\tau$, and an upper bound on the degree $d$, and an upper bound on the integer coefficient lengths.
Output  A supersparse representation of the rational function or FAIL. The algorithm can fail in several ways. The moduli choices may be unlucky, say they cause destructive wrap-around or sparser image fractions. Or the input bounds for degree and sparsity may be too low. Those causes are not distinguishable.
SSRI1 Compute a prime $p_0$ such that $p_0 > 3\tau$.
SSRI2 Compute a list of primes $p_0, p_1, \ldots, p_k$ such that $p_j = m_j 2^j (p_0 - 1) + 1$ and $2^k(p_0 - 1) > 2d$.
SSRI3 Construct a $(p_0 - 1) \times 2(p_0 - 1)$ matrix $A$, where the $i^{th}$ row is $[1, i, i^2, ..., i^{p_0-1}, -f(i), -if(i), -i^2 f(i), ..., -i^{p_0-1} f(i)] \bmod p_0$.
      Do until break
SSRI4   Select at least $p_0$ coefficients and set them to zero. That is, add a row of the form $[0, 0, ..., 0, 1, 0, ..., 0]$, where the 1 corresponds to the coefficient we wish to set to zero.
SSRI5   Compute the nullspace of the above matrix mod $p_0$.
SSRI6   If the number of non-zero entries in a null space vector is no more than $\tau$, record the vector and break from do loop.
SSRI7   Else if there is more than one null space vector, remove at least one more column and go to SSRI5.
SSRI8   Else if all possible combination of columns have been attempted, return FAIL.
      End do
      For $j$ from $1$ to $k - 1$ do
        Let $v^{[num]} = \{e_1^{[num]}, e_2^{[num]}, \ldots, e_{\tau_{[num]}}^{[num]}\}$ and $v^{[den]} = \{e_1^{[den]}, e_2^{[den]}, \ldots, e_{\tau_{[den]}}^{[den]}\}$ be lists containing the exponents of the non-zero terms of the null space vector modulo $2^{j-1}(p_0 - 1)$ of the numerator and denominator, respectively, that were computed in the previous step.
SSRI9   Let $V^{[num]} = \{e, e + 2^{j-1}(p_0 - 1) | e \in v^{[num]}\}$. Let $V^{[den]}$ be computed similarly. (Note that $V^{[num]}$ and $V^{[den]}$ represent the list of possible exponents mod $2^j(p_0 - 1)$.)
SSRI10  Select a list of at least $|V^{[num]}| + |V^{[den]}|$ random residues $i$ mod $p_{j+1}$ and store in $M$, where $|V^{[num]}|$ represents the number of elements in the list $V^{[num]}$.
        Do until break
SSRI11    Construct the $|M| \times (|V^{[num]}| + |V^{[den]}|)$ matrix $A$ where the row for residue $i$ in $M$ is given by $[i^{m_j V_1^{[n]}}, i^{m_j V_2^{[n]}}, ..., -f(i^{m_j}) i^{m_j V_1^{[d]}}, -f(i^{m_j}) i^{m_j V_2^{[d]}}, ...] \bmod p_j$.
SSRI12    Compute the null space of $A$ mod $p$ in reduced column echelon form.
SSRI13    If the null space has dimensions greater than 2, add more residues to $M$ and thus more rows to $A$ and go to SSRI12.
SSRI14    If the null space has dimension 2, select a null space vector at random and break.
SSRI15    If the null space has dimension less than 2, restart algorithm with new initial vector or prime. If all possible starting vectors have been exhausted, return FAIL.
        End do until
SSRI16  Record the exponents from the null space vector in $v^{[num]}$ and $v^{[den]}$. These will be the exponents of the function mod $p_{j+1}$.
      End do
SSRI17 Use rational vector recovery to recover the coefficients of the rational function. Multiply the numerator and denominator by the same power of $x$ so that all exponents are less than the degree bound. (Recall that $2^k(p_0 - 1) > 2d$.) If no such power exists, return DEGREE BOUND TOO SMALL. Otherwise, return the rational function. End of Algorithm.
```

The algorithm inputs bounds on both the number of non-zero terms and the degree. Due to the nature of the linear system, it is of only moderate advantage to have separate degree bounds on the numerator and denominator, because both the numerator and denominator are lifted at every step. However, having separate bounds for the number of non-zero terms in the numerator and denominator can lead to better column elimination strategies in the initial step.

6. COMPLEXITY ANALYSIS

We now attempt to give some plausible reasons that the algorithm can be implemented to work in random polynomial time for a fixed τ, the bound for the sum of the number of terms in a ***sparsest*** solution $g/h \in \mathbb{Q}(x)$, $g, h \in \mathbb{Z}[x]$. We may assume, as before, that both g and h have a factor x.

We do not take into consideration several of the heuristic improvements discussed before, but wish to show that the algorithm can be modified to produce a correct result for ***fixed*** τ. We will work modulo a single large prime $p = \mu q 2^l + 1$, where q is a randomly chosen small prime that needs to have certain properties for the analysis to work, l is sufficiently large to cover the degree bounds, and μ is the smallest multiplier for the arithmetic progression $1 + \lambda\, q 2^l$,

$\lambda = 1, 2, \ldots$ We shall assume that a μ can be found (see Section 4.1). It needs not be small.

The algorithm, once q and l are chosen, computes the coefficients of a sparsest pair $\bar{g}_j, \bar{h}_j \in \mathbb{Z}_p[x]$ in terms of the number of nonzero coefficients of the linear system

$$(g\bar{h}_j - h\bar{g}_j)(x^{m_j}) \equiv 0 \bmod (x^p - x, p), m_j = \mu 2^{l-j}, \qquad (5)$$

for $j = 0$, ..., l. Again, we may assume that $\bar{g}_j$ and $\bar{h}_j$ have a factor x. The terms x^{e+1} in g and h (5) map to $x\, x^{m_j e \bmod (p-1)} = x\, x^{(e \bmod q)\mu 2^{l-j}}$. We shall make the following two sparsity assumptions:

Sp1 q has been sampled so that the term exponents in g and the term exponents in h map to distinct residues modulo q with high probability. For that a random q with $O(\tau^2 \log(\deg g + \deg h))$ suffices: q must not divide the products of differences of the term degrees of the numerator and denominator (cf. [19, Lemma 4.3]).

Sp2 the image of g and h forms a sparsest solution to (5) for all j. At this moment we have no proof that the latter is true with high probability (see Section 2.3).

We shall perform the interpolation in Steps SSRI3 and SSRI-13 above at points γ^η where γ is a primitive root modulo p and η ranges $0 \leq \eta < 2\tau^2$. Primitive roots are abundant as $\varphi(n) = \Omega(n/\ln\ln n)$ and if μ is small can be computed effectively as $p - 1$ is smooth. Since $(g\bar{h}_j - h\bar{g}_j)(x^{m_j}) \bmod (x^p - x)$ has no more than $2\tau^2$ terms, $(g\bar{h}_j - h\bar{g}_j)(\gamma^{\eta m_j}) = 0$ for all η must imply (5) for $\bar{g}_j$, $\bar{h}_j$ by the argument in [21, Section 4.1], as the term evaluations in (5) remain distinct for the primitive root and the corresponding Vandermonde coefficient matrix is of full rank.

In Step SSRI3 our algorithm tests all sparse exponent vectors (modulo q times m_0) for $\bar{g}_0(x^{m_0})$ and $\bar{h}_0(x^{m_0})$. For a sparsest solution, the vector is unique (up to a scalar multiple) by the argument in Section 2.3. Two linearly independent solutions for the same exponent choice can produce a sparser solution via linear combination.

A similar argument works (for some f) for all $j > 0$ in Step SSRI13 above. There are two, possibly linearly independent, sparsest solutions, $\bar{g}_j$, $\bar{h}_j$ and $x^{(p-1)/(2m_j)}\bar{g}_j$, $x^{(p-1)/(2m_j)}\bar{h}_j$ for (5). Note that shifting the second solution by multiplying again by $x^{(p-1)/(2m_j)}$ results in the first since $x\, x^{p-1} \equiv x$. If our assumptions (Sp1) and (Sp2) hold, there is no sparser solution and lifting the sparsest solution corresponding to g and h will lead the sparsest solutions with exactly one of $\hat{\alpha}_i$ and $\tilde{\alpha}_i$ and exactly one of $\hat{\beta}_i$ and $\tilde{\beta}_i$ in (4) equal to 0 for all i. There are still $2^{\tau^{[num]}+\tau^{[den]}}$ many candidate interpolants, each of which we will test. Assume now that there is a third such sparsest interpolant $\bar{\bar{g}}_j, \bar{\bar{h}}_j$ at all γ^η. If the coefficient vector of $\bar{\bar{g}}_j(x^{m_j 2^j}), \bar{\bar{h}}_j(x^{m_j 2^j})$ is not the coefficient vector (or a scalar multiple) of $\bar{g}_0, \bar{h}_0$ then a sparser vector can be constructed for $j = 0$. Otherwise, in the two solutions

$$(\bar{g}_j - \bar{\bar{g}}_j)(x^{m_j 2^j}), (\bar{h}_j - \bar{\bar{h}}_j)(x^{m_j 2^j})$$
$$(x^{(p-1)/(2m_j)}\bar{g}_j - \bar{\bar{g}}_j)(x^{m_j 2^j}), (x^{(p-1)/(2m_j)}\bar{h}_j - \bar{\bar{h}}_j)(x^{m_j 2^j})$$

certain coefficients cancel to 0. In fact, if $\tau^{[num]} + \tau^{[den]}$ is an odd integer, one of the two solutions must have fewer non-zero terms, in contradiction to assumption (Sp2) above.

We add that the exact integer coefficients can be recovered for each j from their residues modulo p and false interpolants may be eliminated by virtue of having large recovered integer coefficients.

7. OBSERVATIONS AND EXPERIMENTS

7.1 Multiple Variables

The algorithm as presented works only for the single variable case. However, a simple variable substitution due to Kronecker can be applied to make this algorithm work in multiple variables. Suppose there are n variables, x_1, ..., x_n and d is a bound on the degree of all individual variables. Then we can apply the substitutions $x_i = x^{(d+1)^{i-1}}$, $i = 1$, ..., n. This means that a general monomial is mapped as follows: $x_1^{e_1} \cdots x_n^{e_n} = x^E$ with $E = e_1 + (d+1)e_2 + \cdots + e_n(d+1)^{n-1}$. Because d is an upper bound on the degree, there is no chance that two multivariate monomials get mapped to the same single variable monomial. Furthermore, since the algorithm is conjectured to be polylogarithmic in the degree, then it would become polynomial in the number of variables.

7.2 Some Experiments

Both algorithms were implemented in Maple 12 and tested on a MacPro with 16 cores (Intel Xeon 2.67GHz) with 32GB of real memory on a Nehalem memory bus running Linux version Ubuntu 2.6.31-22.70 (Ubuntu). The times given in the table are in seconds. The first column is the number of non-zero terms in the numerator and in the denominator, and the first row is the degree of the numerator and denominator. The table below shows the time it took to recover the random rational function with random integer coefficients in the range $-5, \ldots, 5$ and includes the garbage collection cost.

	2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}	2^{70}	2^{100}
3	3.6	4.4	3.9	2.5	7.1	13.5	20.2	729.9
4	3.7	4.2	7.8	6.0	15.0	23.2	51.2	1109.2
5	4.4	5.2	8.2	22.4	30.9	65.9	77.2	809.3
6	5.0	7.5	9.9	27.0	51.6	121.9	278.1	3258.6
7	50.2	38.9	9.8	66.0	193.5	308.4	637.8	2926.5
8	144.7	75.8	339.2	554.4	2247.6	1703.2	8205.8	18092.0
9	1673.8	307.2	289.9	4126.5	2887.2	2011.4	1963.1	45148.1
10	440.9	1951.7	462.3	4453.91	14613.2	6262.2	29445.3	39455.0

The degrees of numerators and denominators given in the first row. The number of non-zero terms in both the numerators and denominators are given in the first column. The second table below shows the number of blackbox calls.

	2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}	2^{70}	2^{100}
3	347	527	707	729	1065	1422	1681	5090
4	377	617	1307	906	1295	1646	2240	5696
5	377	707	1007	1395	1736	2317	2592	5267
6	434	797	1157	1428	1818	2675	3651	7304
7	467	887	1307	1814	2654	3644	4319	7619
8	497	962	1457	2639	4118	5087	9116	12584
9	527	962	2543	3158	4222	5229	6502	11689
10	554	1151	1748	2353	5981	6817	8591	11941

Note that the performance is not uniform due to the sparse vector construction. The first three columns and the 10 terms problem for degree 2^{40} are runs with Silver-Pohlig/ Hellman lifting (Section 4), and the others runs with "easy" lifting (Section 3.2).

Additionally, the algorithm with Kronecker's substitution was used on the rational function $f(x, y) = (x^{2^{100}} - 1) \times (y^{2^{50}} - 1) \;/\; ((x - 1)(y - 1))$. The given sparse form of the function, rather than the very dense reduced form, was recovered with easy lifting in 2771.9 seconds with 8600 black box calls.

7.3 Drawbacks

The algorithm, as presented, has a number of drawbacks. At the moment, there is no proven way of certifying that a null space vector is indeed the correct null space vector. Therefore, it would be possible to lift an incorrect vector, perhaps multiple times, before discovering that the vector does not represent the actual function.

In the lifting step, it is also not known what the dimension of the null space will be. Even if we did know that the matrix has the fullest possible rank given all possible equations, we do not know how many equations are needed to get that rank; however, see Section 6 for a possible solution.

The most significant shortcoming, however, is that the initial guess of the vector appears to be exponential in the number of non-zero terms, since one may have to traverse all possible combinations before the correct vector is found.

8. FURTHER WORK

Again, the obvious weak point of the algorithm is the fact that exponentially many combinations may have to be processed. Although finding the sparsest vector in a null space is NP-hard in general, it is certainly possible that the particular null spaces that are generated have special properties that allow one to find (at least with high probability) a sparse vector in polynomial time. Indeed, in general if one knows one vector in the null space generated in the initial step of the algorithm, one generally knows $p-2$ more vectors in the null space (except for finitely many degenerate p), since if $\frac{f}{g}$ is the rational function that generated the system of equations, then so is $x^i\, f(x)/(x^i\, g(x))$ for $i = 1, \ldots, p-2$. After $p-2$, everything wraps around mod $p-1$.This banded nature of the null space could possibly be exploitable to find sparse vectors in that null space.

If we change the model to one where the black box can be evaluated at complex roots of unity, say modulo cyclotomic polynomials of small degree over the rational numbers, as would be the case if the black box is a straight-line program [5], then the recovery of an initial sparse null space vector with rational entries could be attempted using techniques from compressed sensing, which started with [1]. There are many approaches known today that would recover a sparse vector. The lifting could then still be done modulo larger and larger primes.

Acknowledgments: We thank Éric Schost for many discussions on the supersparse rational function recovery problem, and Carl Pomerance for his references on conjectures related to the effective Dirichlet theorem. The referees have provided many useful comments which we would like to acknowledge.

9. REFERENCES

[1] Candes, E., and Tao, T. Decoding by linear programming. *IEEE Trans. Inf. Theory* IT-*51*, 12 (2005), 4203–4215.

[2] Coleman, T., and Pothen, A. The null space problem I. complexity. *SIAM. J. on Algebraic and Discrete Methods 7* (1986), 527–537.

[3] Cucker, F., Koiran, P., and Smale, S. A polynomial time algorithm for diophantine equations in one variable. *J. Symbolic Comput. 27*, 1 (1999), 21–29.

[4] Filaseta, M., Granville, A., and Schinzel, A. Irreducibility and greatest common divisor algorithms for sparse polynomials, 2007. Manuscript submitted.

[5] Garg, S., and Schost, Éric. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science 410*, 27-29 (2009), 2659 – 2662.

[6] Giesbrecht, M., Kaltofen, E., and Lee, W. Algorithms for computing sparsest shifts of polynomials in power, Chebychev, and Pochhammer bases. *J. Symbolic Comput. 36*, 3–4 (2003), 401–424.

[7] Giesbrecht, M., and Roche, D. S. Interpolation of shifted-lacunary polynomials. *Computing Research Repository abs/0810.5685* (2008). URL: http://arxiv.org/abs/0810.5685.

[8] Giesbrecht, M., and Roche, D. S. On lacunary polynomial perfect powers. In *ISSAC 2008* (New York, N. Y., 2008), D. Jeffrey, Ed., ACM Press, pp. 103–110.

[9] Giesbrecht, M., and Roche, D. S. Detecting lacunary perfect powers and computing their roots. *Computing Research Repository abs/0901.1848* (2009).

[10] Giesbrecht, M., Roche, D. S., and Tilak, H. Computing sparse multiples of polynomials. In *Proc. Internat. Symp. on Algorithms and Computation (ISAAC 2010)* (2010), p. to appear.

[11] Heath-Brown, D. R. Almost-primes in arithmetic progressions and short intervals. *Math. Proc. Camb. Phil. Soc. 83* (1978), 357–375.

[12] Heath-Brown, D. R. Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. *Proc. London Math. Soc 3* (1992), 265–338.

[13] Kaltofen, E. Greatest common divisors of polynomials given by straight-line programs. *J. ACM 35*, 1 (1988), 231–264.

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

[15] Kaltofen, E. Fifteen years after DSC and WLSS2 What parallel computations I do today [Invited lecture at PASCO 2010]. In *PASCO'10 Proc. 2010 Internat. Workshop on Parallel Symbolic Comput.* (New York, N. Y., July 2010), M. Moreno Maza and J.-L. Roch, Eds., ACM, pp. 10–17.

[16] Kaltofen, E., and Koiran, P. Finding small degree factors of multivariate supersparse (lacunary) polynomials over algebraic number fields. In *ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 2006), J.-G. Dumas, Ed., ACM Press, pp. 162–168.

[17] Kaltofen, E., and Lee, W. Early termination in sparse interpolation algorithms. *J. Symbolic Comput. 36*, 3–4 (2003), 365–400. Special issue Internat. Symp. Symbolic Algebraic Comput. (ISSAC 2002). Guest editors: M. Giusti & L. M. Pardo.

[18] Kaltofen, E., and Trager, B. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput. 9*, 3 (1990), 301–320.

[19] Kaltofen, E., and Villard, G. On the complexity of computing determinants. *Computational Complexity 13*, 3-4 (2004), 91–130.

[20] Kaltofen, E., and Yang, Z. On exact and approximate interpolation of sparse rational functions. In *ISSAC 2007 Proc. 2007 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 2007), C. W. Brown, Ed., ACM Press, pp. 203–210.

[21] Kaltofen, E., Yang, Z., and Zhi, L. On probabilistic analysis of randomization in hybrid symbolic-numeric algorithms. In *SNC'07 Proc. 2007 Internat. Workshop on Symbolic-Numeric Comput.* (New York, N. Y., 2007), J. Verschelde and S. M. Watt, Eds., ACM Press, pp. 11–17.

[22] Kipnis, A., and Shamir, A. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Proc. CRYPTO '99* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 19–30.

[23] Lenstra, Jr., H. W. Finding small degree factors of lacunary polynomials. In *Number Theory in Progress* (1999), K. Győry, H. Iwaniec, and J. Urbanowicz, Eds., vol. 1 Diophantine Problems and Polynomials, Stefan Banach Internat. Center, Walter de Gruyter Berlin/New York, pp. 267–276.

[24] Mikawa, H. On primes in arithmetic progressions. *Tsukuba J. Mathematics 25*, 1 (2001), 121–153.

[25] Olesh, Z., and Storjohann, A. The vector rational function reconstruction problems. In *Proc. Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov (WWCA)* (2007), pp. 137–149.

[26] Plaisted, D. A. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theoretical Comput. Sci. 13* (1984), 125–138.

[27] Pohlig, C. P., and Hellman, M. E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inf. Theory* IT-*24* (1978), 106–110.