Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields

Joris van der Hoeven, Grégoire Lecerf

▶ **To cite this version:**

Joris van der Hoeven, Grégoire Lecerf. Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields. 2019. hal-02382117v1

HAL Id: hal-02382117

https://hal.science/hal-02382117v1

Preprint submitted on 27 Nov 2019 (v1), last revised 1 May 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sparse polynomial interpolation

Exploring fast heuristic algorithms over finite fields

JORIS VAN DER HOEVEN[abc], GRÉGOIRE LECERF[bd]

[a]. CNRS (UMI 3069, PIMS)
Department of Mathematics
Simon Fraser University
8888 University Drive
Burnaby, British Columbia
V5A 1S6, Canada

[b]. CNRS, École polytechnique, Institut Polytechnique de Paris
Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161)
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France

[c]. *Email:* vdhoeven@lix.polytechnique.fr
[d]. *Email:* lecerf@lix.polytechnique.fr

Work in progress, draft version of November 27, 2019

Consider a multivariate polynomial $f \in K[x_1, \ldots, x_n]$ over a field K, which is given through a black box capable of evaluating f at points in K^n, or possibly at points in A^n for any K-algebra A. The problem of sparse interpolation is to express f in its usual form with respect to the monomial basis. We analyze the complexity of various old and new algorithms for this task in terms of bounds D and T for the total degree of f and its number of terms. We mainly focus on the case when K is a finite field and explore possible speed-ups under suitable heuristic assumptions.

1. INTRODUCTION

Consider a polynomial function $f: K^n \to K$ over a field K given through a black box capable of evaluating f at points in K^n. The problem of sparse interpolation is to recover the representation of $f \in K[x_1, \ldots, x_n]$ in its usual form, as a linear combination

$$f = \sum_{i_1, \ldots, i_n} f_{i_1, \ldots, i_n} x_1^{i_1} \cdots x_n^{i_n} \tag{1}$$

of monomials. The aim of this paper is to analyze various approaches for solving this problem, with our primary focus on the case when K is a finite field. We will survey and synthesize known algorithms, but we will also present a few new algorithms, together with improved complexity bounds for some important special cases.

> *We explore various methods under heuristic conditions that we expect to fairly reflect average behavior in practice. We preferred a relaxed and intuitive style of exposition to mathematically precise theorems with rigorous proofs.*

Efficient algorithms for the task of sparse interpolation go back as far as to the eighteen's century and the work of Prony [41]. The first modern version of the algorithm is due to Ben Or and Tiwari [8]. This method was swiftly embraced in computer algebra [11, 28, 32, 34, 36, 37, 39]; for early implementations, we refer to [13, 14]. There has been a regain of interest for the problem during the last decade, both from a theoretical perspective [2, 3, 4, 15, 16, 29, 30, 33] and from the practical point of view [25, 26, 27, 31, 35]. We also mention the survey paper [43] by Roche on the more general topic of computations with sparse polynomials.

1.1. Complexity considerations

Throughout this paper d will stand for the total degree of f and t for the number of non-zero terms in (1). Whenever available, the uppercase characters D and T represent upper bounds for d and t. We will also write L for the number of ring or field operations in K that are required in order to evaluate f.

The complexity analysis of sparse interpolation has to be carried out with a lot of care, due to the large variety of cases that can occur:

- What kind of complexity/evaluation model do we use?
  - Do we count the number operations in K or the number of bit operations?
  - Are we interested in theoretic (asymptotic) or practical complexity?
  - Are divisions allowed for the evaluation of f and how do we count them?
  - Are we only allowed to evaluate f at points in K^n or also at points in $\hat{K}^n$ for certain extension rings or fields $\hat{K}$?
- What kind of coefficient field K do we use?
  - A field from analysis such as $\mathbb{C}$.
  - A discrete field such as $\mathbb{Q}$ or a finite field $\mathbb{F}_q$.
  - Fields with roots of unity ω of large smooth order in K.
- The univariate case ($n=1$) *versus* the multivariate case ($n>1$).
- Informally speaking, there are three levels of "sparsity":
  - Weakly sparse: total degrees d of the order $O(\log t)$.
  - Normally sparse: total degrees d of the order $t^{O(1)}$.
  - Super sparse: total degrees of order d with $\log t = o(\log d)$.

We also notice that almost all general algorithms for sparse interpolation are probabilistic of Monte Carlo type. Indeed, without further *a priori* knowledge about f, such as its support or its number of terms, the mere knowledge of a finite number of evaluations of f only allow us to guess plausible expressions for f.

In this paper, we will be mostly interested in the practical bit complexity of sparse interpolation over finite fields $K=\mathbb{F}_q$. Sparse interpolation over the rational numbers can often be reduced to this case as well, in which case $q=p$ is a well chosen prime number that fits into 32 or 64 bits and such that $p-1$ admits a large smooth divisor; see section 6.5. We analyze the complexities of specializations of existing algorithms to the finite field case and also present a few new algorithms and tricks for this specific setting. Due to the large number of cases that can occur, we will not prove detailed complexity bounds for every single case, but rather outline how various ideas may be used and combined to reduce the practical complexity.

From our practical perspective, it is important to take into account logarithmic factors in complexity bounds, but it will be convenient to ignore sublogarithmic factors. For this reason, we use the *ad hoc* notation

$$\varphi = O^\flat(\psi) \stackrel{\text{def}}{\iff} \varphi = O(\psi\,(\log\psi)^{o(1)}\,(\log\log\,(TDq))^{O(1)})$$

for any functions φ, ψ.

We will also write $\mathsf{M}_K(d)$ for the bit cost of multiplying two polynomials of degree d over K and abbreviate $\mathsf{M}_q(d) := \mathsf{M}_{\mathbb{F}_q}(d)$. For instance, the naive multiplication algorithm yields $\mathsf{M}_q(d) = O^\flat(d^2 \log^2 q)$. For our complexity analyses we will give priority to the asymptotic complexity point of view and use the well known [20, 44] bound $\mathsf{M}_q(d) = O^\flat(d \log d \log q)$.

1.2. Overview of the paper

Many of the challenges concerning sparse interpolation already arise in the univariate case when $n = 1$. As we will see in section 7.1, the multivariate case can actually be reduced to the univariate one using the technique called "Kronecker segmentation", even though other approaches may be more efficient. For this reason, a large part of the paper is devoted to methods for interpolating a univariate black box function $f(x)$.

We distinguish three major types of algorithms:

- Cyclic extension methods (section 4).
- Geometric progression methods (section 5).
- FFT based methods (section 6).

For the first two types of methods we mostly review existing algorithms, although we do propose some new variants and optimizations. The third FFT based method is new, as far as we are aware. For each of the three methods, an important *leitmotif* is to evaluate $f(x)$ modulo $x^r - 1$ for one or more suitable orders r, after which we reconstruct $f(x)$ from these modular projections.

Cyclic extension methods directly evaluate f over the cyclic extension ring $K[x]/(x^r - 1)$. This has the advantage that r can be freely chosen in a suitable range. However, the evaluation of f over such a large cyclic extension induces a non-trivial overhead in the dependency of the complexity on L.

Geometric progression methods rather evaluate f at a sequence $1, \omega, \ldots, \omega^{2T-1}$ of pairwise distinct elements in K (or inside an extension of K of modest degree s). If $K = \mathbb{F}_q$ is a finite field, then ω necessarily has an order r that divides $q-1$ (or $q^s - 1$ when working over an extension of degree s). Although the evaluations of f become more efficient using this approach, the recovery of $f(x)$ modulo $x^r - 1$ from $f(1), f(\omega), \ldots, f(\omega^{2T-1})$ requires extra work. The cost of this extra work highly depends on the kind of orders r that can be taken as divisors of $q-1$ (or $q^s - 1$ for small s). Theoretically speaking, the existence of suitable orders r is a delicate issue; in practice, they always tend to exist as long as $D = T^{O(1)}$; see sections 2, 6.3 and 6.4 for some empirical evidence.

Geometric progression methods allow us to take r much larger than T, but they involve a non-trivial cost for recovering $f(x)$ modulo $x^r - 1$ from $f(1), f(\omega), \ldots, f(\omega^{2T-1})$. If $L = o((\log T)^3)$, then this cost may even dominate the cost of the evaluations of f. In such situations, an alternative approach is to evaluate f at $1, \omega, \ldots, \omega^{r-1}$ and to recover $f(x)$ modulo $x^r - 1$ using one inverse DFT of length r. However, this puts an even larger strain on the choice of r, since it is important that $T \leqslant r = O(T)$ for this approach to be

Method	Complexity bound	Conditions
Cyclic extension	$O^\flat(LT\log D\log(qT))$	**H1** and $\log D = O(T)$
Geometric progression	$O^\flat\Big((L+(\log T)^3)\,T\left(\frac{\log D}{\log T}\right)^3\log(qT)\Big)$	**H1**, **H2**, and $D = T^{O(1)}$
FFT	$O^\flat\Big((L+\log T)\,T\left(\frac{\log D}{\log T}\right)^2\log(qT)\Big)$	**H1**, **H2**, and $D = T^{O(1)}$

Table 1. Heuristic complexity bounds for the three main approaches to sparse interpolation.

efficient. Moreover, the recovery of $f(x)$ from its reductions modulo x^r-1 for several orders r of this type is more delicate and based on a probabilistic analysis. Yet, suitable orders r again tend to exist in practice as long as $D = T^{O(1)}$.

The expected complexities of the best versions of the three approaches are summarized in Table 1. These bounds rely on two types of heuristics:

H1. For $r \ll d$, the exponents of f are essentially randomly distributed modulo r.

H2. For $s = O(\log D/\log T)$, the number q^s-1 admits a large smooth divisor.

We will present a more precise version of **H1** in section 4. The heuristic **H2** will be made precise in sections 5.6 and 6.2 and numeric evidence is provided in sections 2, 6.3 and 6.4.

The last section 7 is dedicated to the interpolation of multivariate polynomials. We start with the well known strategies based on Kronecker segmentation (section 7.1) and prime factorization (section 7.2). For sparse polynomials in many variables, but with a modest total degree d, we also recall the inductive approach by packets of coordinates in section 7.3. If $T < q$, then geometric progression and FFT based methods should be particularly favorable in combination with this inductive approach, since one can often avoid working over extensions of K in this case. We conclude section 7 with a few algorithms for special situations.

2. PRELIMINARIES ON FINITE FIELDS

One remarkable feature of the finite field $\mathbb{F}_q$ with q elements is that every $a \in \mathbb{F}_q$ satisfies the equation $a^q = a$. In particular, for any sparse polynomial f as in (1) and with coefficients in $\mathbb{F}_q$, the polynomial f takes the same values as

$$\sum_{i_1,\ldots,i_n} f_{i_1,\ldots,i_n} x_1^{i_1 \operatorname{rem}(q-1)} \cdots x_n^{i_n \operatorname{rem}(q-1)},$$

for $x_1,\ldots,x_n \in \mathbb{F}_q$, where "rem" stands for the remainder of a Euclidean division. In other words, the exponents of f are only determined modulo $q-1$, so we may assume without loss of generality that they all lie in $\{0,\ldots,q-2\}$ and that the total degree d of f satisfies $d \leqslant n\,(q-2)$.

On the other hand, in the case that our black box function f can be evaluated not only over $\mathbb{F}_q$, but also over field extensions $\mathbb{F}_{q^s}$ (this is typically the case if f is given by an expression or a directed acyclic graph (dag)), then the exponents in the expression (1) can be general non-negative integers, but the above remark shows that we will crucially need to evaluate over extensions fields $\mathbb{F}_{q^s}$ with $s > 1$ in order to recover exponents that exceed q.

More generally, if we choose to evaluate f only at points $(a_1,\ldots,a_n) \in \mathbb{F}_{q^s}$ such that $a_1,\ldots,a_n$ are r-th roots of unity, then we can only hope to determine the exponents modulo r in the expansion (1). In that case, r must divide the order q^s-1 of the multiplicative group

s	2^s-1	3^s-1	5^s-1
1		2	2^2
2	3	2^3	$2^3\cdot 3$
3	7	$2\cdot 13$	$2^2\cdot 31$
4	$3\cdot 5$	$2^4\cdot 5$	$2^4\cdot 3\cdot 13$
6	$3^2\cdot 7$	$2^3\cdot 7\cdot 13$	$2^3\cdot 3^2\cdot 7\cdot 31$
8	$3\cdot 5\cdot 17$	$2^5\cdot 5\cdot 41$	$2^5\cdot 3\cdot 13\cdot 313$
12	$3^2\cdot 5\cdot 7\cdot 13$	$2^4\cdot 5\cdot 7\cdot 13\cdot 73$	$2^4\cdot 3^2\cdot 7\cdot 13\cdot 31\cdot 601$
16	$3\cdot 5\cdot 17\cdot 257$	$2^6\cdot 5\cdot 17\cdot 41\cdot 193$	$2^6\cdot 3\cdot 13\cdot 17\cdot 313\cdot 11489$
24	$3^2\cdot 5\cdot 7\cdot 13\cdot 17\cdot 241$	$2^5\cdot 5\cdot 7\cdot 13\cdot 41\cdot 73\cdot 6481$	$2^5\cdot 3^2\cdot 7\cdot 13\cdot 31\cdot 313\cdot 601\cdot 390001$
30	$3^2\cdot 7\cdot 11\cdot 31\cdot 151\cdot 331$	$2^3\cdot 7\cdot 11^2\cdot 13\cdot 31\cdot 61\cdot 271\cdot 4561$	$2^3\cdot 3^2\cdot 7\cdot 11\cdot 31\cdot 61\cdot 71\cdot 181\cdot 521\cdots$
36	$3^3\cdot 5\cdot 7\cdot 13\cdot 19\cdot 37\cdot 73\cdot 109$	$2^4\cdot 5\cdot 7\cdot 13\cdot 19\cdot 37\cdot 73\cdot 757\cdot 530713$	$2^4\cdot 3^3\cdot 7\cdot 13\cdot 19\cdot 31\cdot 37\cdot 601\cdot 829\cdots$

Table 2. Prime factorizations of 2^s-1, 3^s-1, and 5^s-1 for various small smooth values of s.

s	1299743^s-1	$a(s)$
1	$2\cdot 649871$	2
2	$2^6\cdot 3^2\cdot 4513\cdot 649871$	$2^3\cdot 3$
3	$2\cdot 7\cdot 13\cdot 397\cdot 649871\cdot 6680137$	2
4	$2^7\cdot 3^2\cdot 5^2\cdot 193\cdot 4349\cdot 4513\cdot 40253\cdot 649871$	$2^4\cdot 3\cdot 5$
6	$2^6\cdot 3^3\cdot 7^2\cdot 13\cdot 31\cdot 397\cdot 4513\cdot 649871\cdot 6680137\cdot 18164844799$	$2^3\cdot 3^2\cdot 7$
8	$2^8\cdot 3^2\cdot 5^2\cdot 17^2\cdot 73\cdot 193\cdot 241\cdot 4349\cdot 4513\cdot 40253\cdot 649871\cdot 298090889\cdot 941485217$	$2^5\cdot 3\cdot 5$
12	$2^7\cdot 3^3\cdot 7^2\cdot 13\cdot 31\cdot 193\cdot 397\cdot 4349\cdot 4513\cdot 40253\cdot 649871\cdot 6680137\cdot 387205657\cdots$	$2^4\cdot 3^2\cdot 5\cdot 7\cdot 13$
16	$2^9\cdot 3^2\cdot 5^2\cdot 17^2\cdot 73\cdot 193\cdot 241\cdot 4349\cdot 4513\cdot 40253\cdot 649871\cdot 3955153\cdot 298090889\cdots$	$2^6\cdot 3\cdot 5\cdot 17$
24	$2^8\cdot 3^3\cdot 5^2\cdot 7^2\cdot 13\cdot 17^2\cdot 31\cdot 73\cdot 193\cdot 241\cdot 397\cdot 4349\cdot 4513\cdot 40253\cdot 649871\cdots$	$2^5\cdot 3^2\cdot 5\cdot 7\cdot 13$
30	$2^6\cdot 3^3\cdot 7^2\cdot 11\cdot 13\cdot 31\cdot 61\cdot 71\cdot 271\cdot 397\cdot 701\cdot 881\cdot 1171\cdot 2411\cdot 4513\cdot 649871\cdots$	$2^3\cdot 3^2\cdot 7\cdot 11\cdot 31$
36	$2^7\cdot 3^4\cdot 5^2\cdot 7^2\cdot 13\cdot 31\cdot 37\cdot 109\cdot 193\cdot 397\cdot 757\cdot 1657\cdot 4349\cdot 4513\cdot 40253\cdot 649871\cdots$	$2^4\cdot 3^3\cdot 5\cdot 7\cdot 13\cdot 19\cdot 37$

Table 3. Prime factorizations of q^s-1 and $a(s)$ for $q=1299742$ and various values of s.

of $\mathbb{F}_{q^s}$. In addition, as we will recall in sections 5.1 and 5.2 below, several important tools such as polynomial root finding and discrete logarithms admit faster implementations if we can take r of the form $r=r_1r_2$ with $r_1=O(T)$ and where r_2 is smooth. Sometimes, primitive roots of unity of such orders r already exist in $\mathbb{F}_q$. If not, then we need to search them in extension fields $\mathbb{F}_{q^s}$ with $s>1$ as small as possible.

Let us briefly investigate the prime factorization of q^s-1 for various q and small s. As observed in [21], the number q^s-1 typically admits many small prime divisors when s is itself smooth. This phenomenon is illustrated in Table 2 for small values of q. For practical purposes, given T, it is easy in practice to find a small s and divisors $r=r_1r_2\mid(q^s-1)$ such that $r_1\approx T$ and r_2 is smooth.

For larger q, we may need to resort to larger extension degrees s in order to find appropriate orders r. A natural question is whether q^s-1 is guaranteed to have a non-trivial smooth divisor for large q and a fixed value of s. This leads us to introduce the following guaranteed lower bound:

$$a(s) := \lim_{q_0\to\infty} \gcd\,(q^s-1 : q\geqslant q_0). \tag{2}$$

In Table 3, we have shown the prime factorizations of q^s-1 and $a(s)$ for $q=1299743$ and various small smooth values of s. Here $q=1299743$ was chosen such that $(q-1)/2$ is also prime. For the practical applications in this paper, the table suggests that it remains likely that suitable orders r can still be found whenever needed, and that $a(s)$ is usually quite pessimistic, even for large values of q. Let us finally mention that the sequence $a(s)$ coincides with Sloane's integer sequence A079612; see https://oeis.org/A079612.

3. GENERAL OBSERVATIONS

As already mentioned in the introduction, most algorithms for sparse interpolation are probabilistic of Monte Carlo type. We notice that it is easy to check (with high probability) whether a candidate sparse interpolation f^* of f is correct. Indeed, it suffices to evaluate $f - f^*$ at random sample points and check whether the result vanishes. Deterministic algorithms exist but with higher complexities; see for instance [22, 42].

Many algorithms for sparse interpolation require extra information, such as bounds $T \geqslant t$ and $D \geqslant d$ for the number of terms and the total degree of f. Furthermore, several algorithms are only able to guess some of the terms of f with high probability, but not all of them. In this section, using ideas from [3], we show how to turn such "partial" algorithms for sparse interpolation into full-blown ones. Provided that the characteristic of K is zero or sufficiently large, we also show how to upgrade an interpolation method modulo $(x_1^r - 1, \ldots, x_n^r - 1)$ into a general algorithm, following [30].

3.1. From partial to full interpolation

Assume that we have an algorithm for "partial" sparse interpolation that takes a black box for f as input, together with bounds T and D for t and d. The algorithm should always return a sparse polynomial f^* of total degree at most D and with at most T terms. Moreover, for some fixed constant $0 < \lambda < 1$, if $t \leqslant T$ and $d \leqslant D$, then $f - f^*$ should contain at most λt terms, with high probability. If $t > T$ or $d > D$, then we allow f^* to be essentially arbitrary under the above constraint that f^* has at most T terms of degree $\leqslant D$. Then we may use the following strategy for arbitrary sparse interpolations:

Algorithm 1
Input: a polynomial black box function $f(x_1, \ldots, x_n)$
Output: the sparse interpolation f^* of f

1. Let $f^* := 0$ be an initial approximation of f.
2. Set initial bounds $T := 1$ and $D := 1$ for the number of terms and total degree.
3. Determine the approximate sparse interpolation δ^* to $\delta := f - f^*$ using T and D as bounds for the number of terms and the total degree of δ.
4. Set $f^* := f^* + \delta^*$.
5. If $f = f^*$ with high probability, then return f^*.
6. Whenever appropriate, increase T and/or D, and reset $f^* := 0$.
7. Return to step 3.

In step 6, the precise policy for increasing T and D may depend on the application. We typically double T when t is suspected to exceed T and we double $\log D$ when d is suspected to exceed D. In this way, the bounds T and $\log D$ are at most twice as large as the actual values t and $\log d$, and the running time is essentially a constant times the running time of the approximate sparse interpolation with bounds T and D.

However, for this "meta complexity bound" to hold, it is crucial in step 3 that the sparse approximation f^* can be evaluated efficiently at the sample points used during the sparse interpolation (the naive evaluation of a polynomial with t terms at $\Theta(t)$ points would take time $\Theta(t^2)$, which is too much). Fortunately, this is the case for all sparse interpolation strategies that will be considered in this paper.

When do we suspect T or D to be too low in step 6? In the case of T, a natural idea is to test whether the number of terms in f^* or δ^* exceeds a fixed constant portion of T. This strategy assumes that δ^* be essentially random when T is too small (if the number of terms of δ^* is much smaller than T whenever $t > T$, then we might need more than $\Theta(\log t)$ iterations before the algorithm converges).

The handling of exponents and degree bounds is more involved. When interpolating over a finite field, all non-zero evaluation points are roots of unity, so the exponents can only be determined modulo a certain integer r (or even modulo a submodule of $\mathbb{Z}^n$). If the characteristic of K is sufficiently large, then the exponents can be computed directly: see the next subsection. Otherwise, we need to recombine reductions with respect to several moduli: see section 4 below. This also provides a natural test in order to check whether $d \leqslant D$. Indeed, it suffices to compute the sparse interpolation of f for one or more extra moduli and check whether the results agree with our candidate interpolation.

3.2. Supersparse interpolation in large characteristic

Assume that we have an algorithm that allows us to compute the sparse interpolation of f modulo $I_r = (x_1^r - 1, \ldots, x_n^r - 1)$ for given moduli r. Assume also that we have access to the program that computes f, typically in terms of a straight-line program. If $\operatorname{char} K = 0$ or $\operatorname{char} K > \max(\deg_{x_1} f, \ldots, \deg_{x_n} f)$, then let us show how to derive an algorithm for the sparse evaluation of f.

It is well known that Baur–Strassen's technique of "automatic differentiation" [7] allows us to evaluate the gradient $(\partial f / \partial x_1, \ldots, \partial f / \partial x_n)$ using at most $4L$ operations in K. Using $5L + n$ operations, this provides an algorithm for the simultaneously evaluation of $f, g_1, \ldots, g_n$ with $g_i = x_i (\partial f / \partial x_i)$ for $i = 1, \ldots, n$. With a small overhead, this next allows us to jointly compute the sparse interpolations of $f, g_1, \ldots, g_n$ modulo I_r.

Now assume that $c x_1^{e_1} \cdots x_n^{e_n}$ is a term of f such that for any other term $c' x_1^{e'_1} \cdots x_n^{e'_n}$ of f, we have $(e_1 \bmod r, \ldots, e_n \bmod r) \neq (e'_1 \bmod r, \ldots, e'_n \bmod r)$; we say that this term "does not collide" modulo r. Then the sparse interpolation of f modulo I_r contains the non-zero term $c x_1^{e_1 \bmod r} \cdots x_n^{e_n \bmod r}$. Moreover, given $i \in \{1, \ldots, n\}$ with $e_i \neq 0$, the sparse interpolation of g_i modulo I_r also contains the non-zero term $e_i c\, x_1^{e_1 \bmod r} \cdots x_n^{e_n \bmod r}$. This allows us to retrieve e_i through one simple division $(e_i c) / c$.

Furthermore, if the modulus r was picked at random with $r > t$, then there is a high probability that a fixed non-zero proportion of terms in f do not collide modulo r. Combined with Algorithm 1, this yields an algorithm for obtaining the sparse interpolation of f. This strategy for sparse interpolation was first exploited by Huang [30].

Remark. For simplicity, we consider sparse interpolation of polynomials over fields K in this paper. In fact, the algorithms also work for vectors of polynomials in $K[x_1, \ldots, x_n]^\nu$, by considering them as polynomials with coefficients in K^ν. We implicitly used this fact above when saying that we "jointly" compute the sparse interpolation of $f, g_1, \ldots, g_n$ modulo I_r.

3.3. Conclusion

In summary, we have shown how to reduce the general problem of sparse interpolation to the case when

1. we have bounds for the number of terms and the total degree, and
2. we only require an approximate sparse interpolation (in the sense of section 3.1).

4. UNIVARIATE INTERPOLATION USING CYCLIC EXTENSIONS

One general approach for sparse interpolation of univariate polynomials over general base fields K was initiated by Garg and Schost [15]. It assumes that the black box function f can be evaluated over any cyclic extension of the form $K[x]/(x^r-1)$. The evaluation of

$$f = c_1 x^{e_1} + \cdots + c_t x^{e_t} \tag{3}$$

at x inside such an extension simply yields

$$f^{[r]} := c_1 x^{e_1 \operatorname{rem} r} + \cdots + c_t x^{e_t \operatorname{rem} r}.$$

In absence of "collisions" $e_i = e_j$ modulo r for $i \neq j$, this both yields the coefficients of f and its exponents modulo r. By combining the evaluations for various moduli, it is possible to reconstruct the actual exponents using Chinese remaindering.

Throughout this section, we assume that we are given bounds D and T for the degree d and the number of terms t of f. Garg and Schost's original algorithm was deterministic under these assumptions. However, their algorithm was not designed to be efficient in practice. In the past decade, many variants have been proposed. Roughly speaking, they all follow the same recipe that we summarized in Algorithm 2 below. The variants mainly differ in the precise way recombinations are done in step 3.

Algorithm 2
Input: a black box polynomial $f(x)$, a degree bound D, a sparsity bound T
Output: a partial sparse interpolation f^* of f

1. Determine suitable moduli $r_1, \ldots, r_l > T$ with $\operatorname{lcm}(r_1, \ldots, r_l) > D$.
2. Evaluate f at x in $K[x]/(x^{r_i}-1)$ for $i = 1, \ldots, l$.
3. Determine matching terms in the expansions of $f^{[r_1]}, \ldots, f^{[r_l]}$ that are likely to be the reductions of the same term $c_i x^{e_i}$ in the expansion of f.
4. Return the sum f^* of all terms $c_i x^{e_i}$ as above.

4.1. Complexity analysis

For all matching strategies that have been considered so far, the cost of steps 1 and 3 is dominated by the cost of step 2. If the evaluation of f only involves ring operations, then the running time of Algorithm 2 is therefore bounded by $O((\mathsf{M}_K(r_1) + \cdots + \mathsf{M}_K(r_l))\, L)$. The moduli r_k are usually all of the same order of magnitude $r_k \approx T^\nu$ for some small $\nu \geqslant 1$ that depends on the matching strategy. Then we may take $l = O(\log D / \log T)$, so the cost simplifies to $O(L\, \mathsf{M}_K(T^\nu) \log D / \log T)$. For finite fields $K = \mathbb{F}_q$, this cost becomes $O^\flat(L T^\nu \log D \log q)$. For the design of matching strategies, it is therefore important that we can take ν as small as possible.

Remark. The above analysis can be refined by maintaining separate counts L_{add}, L_{mul}, and L_{div} for the numbers of additions (or subtraction), multiplications, and divisions that are necessary for one evaluation of T. Then the cost of Algorithm 2 over $\mathbb{F}_q$ becomes $O^\flat(T^\nu \log q\, (L_{\text{add}} + L_{\text{mul}} \log T + L_{\text{div}} (\log T)^2))$.

Remark. The complexity analysis may need to be adjusted somewhat if D is so large that we run out of suitable moduli r_i. If our matching strategy requires prime numbers of the order of T^c, then this happens when $\log D$ exceeds approximately the same order T^c. In that case, we need to replace T by an appropriate power of $\log D$ in our complexity bounds. Alternatively, if the characteristic of K is sufficiently large, then we may fall back on the strategy from section 3.2.

4.2. Survey of existing variants based on cyclic extensions

4.2.1. Determining the exponents using Chinese remaindering

Garg and Schost's original algorithm from [15] uses prime numbers p for the moduli r_k. Assuming that $f^{[p]}$ admits t terms, the algorithm is based on the observation that the projection of the polynomial $(z-e_1)\cdots(z-e_t)$ modulo p coincides with $(z-e_1 \operatorname{rem} p)\cdots(z-e_t \bmod p)$. This allows for the recovery of $E=(z-e_1)\cdots(z-e_t)$ through Chinese remaindering, by working with a sufficient number of primes. It then suffices to determine the zeros $e_1,\ldots,e_t$ of E and to recover c_i as the coefficient of $x^{e_i \operatorname{rem} p}$ in $f^{[p]}$ for $i=1,\ldots,t$.

However, this strategy is very sensitive to collisions, and requires $p \gg T^2$ in order to ensure with high probability that $f^{[p]}$ admits exactly t terms. In other words, it forces us to take $\nu \geqslant 2$ in the complexity analysis. Garg and Schost's original algorithm is actually deterministic and uses $\tilde{O}(L\,T^4\,(\log D)^2\,)$ operations in K. The derandomization is achieved by using $\Theta(T^2 \log D)$ different primes p.

4.2.2. Composite moduli

Another matching strategy for step 3 of Algorithm 2 has been proposed by Arnold, Giesbrecht, and Roche [3]. The idea is to pick $r_k = p_0 p_k$ for $k=1,\ldots,l$, where $p_0,\ldots,p_l$ are primes with $p_0 \asymp T$ and $p_1,\ldots,p_l \approx T^\epsilon$ for some $\epsilon > 0$ (so that $\nu = 1+\epsilon$). For $i=1,\ldots,t$ and $k=1,\ldots,l$, there then exists a fixed non-zero probability such that the term $c_i\,x^{e_i \operatorname{rem} p_0}$ of $f^{[p_0]}$ matches a term $c_i\,x^{e_i \operatorname{rem} r_k}$ of $f^{[r_k]}$. Let $\mathcal{K}_i$ be the set of indices k for which we have a match. For some fixed constant $c>0$, we then have $\operatorname{lcm}(r_k : k \in \mathcal{K}_i) > T^{c\epsilon l}$ with high probability. By taking $l > \log D / (c\,\epsilon \log T)$ in step 1, this implies $\operatorname{lcm}(r_k : k \in \mathcal{K}_i) > D$. With high probability, this allows us to reconstruct those terms $c_i x^{e_i}$ such that $e_i \neq e_j$ modulo p_0 for all $j \neq i$. The sum of these terms gives the desired approximation f^* of f for which a fixed non-zero proportion of terms are likely to be correct.

4.2.3. Diversification

Giesbrecht and Roche proposed yet another matching strategy [16] which is based on the concept of "diversification". The polynomial f is said to be *diversified* if its coefficients c_i are pairwise distinct. Assuming that K is sufficiently large, it is shown in [16] that the polynomial $f(\alpha\,x)$ is diversified with high probability for a random choice of $\alpha \in K^*$. Without loss of generality, this allows us to assume that f is diversified.

In step 3 of Algorithm 2, we then match a term $c\,x^e$ of $f^{[r_k]}$ with a term $c'\,x^{e'}$ of $f^{[r_{k'}]}$ if and only if $c=c'$. Giesbrecht and Roche's original algorithm uses $l = O(\log D)$ moduli $r_1,\ldots,r_l$ of size $\tilde{O}(T^2 \log D)$. Consequently, their algorithm for sparse interpolation uses $\tilde{O}(L\,T^2\,(\log D)^2)$ operations in $\mathbb{F}_q$. As we will see below, their probabilistic analysis is actually quite pessimistic: in practice, it is possible to take $r_1,\ldots,r_l = \tilde{O}(T)$ as long as $\log D = O(T)$.

4.3. An optimized probabilistic algorithm based on diversification

Let us now focus on the design and analysis of a probabilistic algorithm that exploits the idea of diversification even more than Giesbrecht and Roche's original method from [16].

Given a diversified polynomial f, together with bounds D and T for its degree d and its number of terms t, our aim is to compute $\Theta(t)$ terms of f, with high probability. Our algorithm uses the following parameters:

- A constant $\tau \asymp 1$.
- Orders $r_1 < \cdots < r_l$ that are pairwise coprime, with $r_k \approx \tau T$ for $k = 1, \ldots, l$.
- The minimal number $\nu \in \{1, \ldots, l\}$ such that $\operatorname{lcm}(r_1, \ldots, r_\nu) = r_1 \cdots r_\nu > D$.

The precise choice of τ and l will be detailed below; the parameter τ and the ratio l/ν should be sufficiently small for the algorithm to be efficient, but τ and l/ν should also be sufficiently large for our algorithm to succeed with high probability.

We now use Algorithm 3 below in order to compute an approximate sparse interpolation of f. It is a variant of Algorithm 2 with a matching strategy that has been detailed in steps 2, 3, and 4. Each individual term $c x^e$ of f is reconstructed from only a subset of its reductions modulo $x^{r_1} - 1, \ldots, x^{r_l} - 1$.

Algorithm 3
Input: a diversified black box polynomial $f(x)$ and parameters as above
Output: an approximate sparse interpolation f^* of f

---

1. Compute $f^{[r_k]} = f \operatorname{rem} (x^{r_k} - 1)$ for $k = 1, \ldots, l$.
2. Let $f^* := 0$.
3. Let C_k be the set of all coefficients that occur once in $f^{[r_k]}$, for $k = 1, \ldots, l$.
4. For each $c \in C_1 \cup \cdots \cup C_l$ do:
   If $\mathcal{K} := \{k : c \in C_k\}$ is such that $\operatorname{lcm}(r_k : k \in \mathcal{K}) > D$, then:
   Determine the unique exponent $e < D$ such that $c x^{e \operatorname{rem} r_k}$ occurs in $f^{[r_k]}$ for every $k \in \mathcal{K}$, and set $f^* := f^* + c x^e$.
5. Return f^*.

4.4. Analysis of the expected number of correct terms

How to ensure that a non-zero portion of the terms of f^* can be expected to be correct? In order to answer this question, we make the following heuristic hypothesis:

$\mathbf{H_{red}}$. For $k = 1, \ldots, l$, the modular reductions of exponents $e_i \operatorname{rem} r_k$ for $i = 1, \ldots, t$ are uniformly distributed in $\{0, \ldots, r_k - 1\}$. The distribution associated to r_k is independent of the one associated to $r_{k'}$ whenever $k \neq k'$.

Such a heuristic is customary in computer science, typically when using hash tables.

According to $\mathbf{H_{red}}$, the probability that a fixed term $c x^e$ does not collide with another term modulo r_k is

$$(1 - r_k^{-1})^{T-1} \approx (1 - (\tau T)^{-1})^{T-1} \geqslant (1 - (\tau T)^{-1})^T \geqslant \mathrm{e}^{T \log(1 - (\tau T)^{-1})} \geqslant \mathrm{e}^{-1/\tau}.$$

Setting $\epsilon := 1 - \mathrm{e}^{-1/\tau}$, this probability tends to $1 - \epsilon$ for large T. The probability that $c x^e$ collides with another term modulo r_k for exactly κ values of k is therefore bounded by

$$\binom{l}{\kappa} \epsilon^{l-\kappa} (1 - \epsilon)^\kappa$$

and this bound is sharp for large T. Consequently, the probability that we cannot recover a term $f_{e_i} x^{e_i}$ in step 4 from its reductions modulo $x^{r_k}-1$ for $k=1,\ldots,l$ is at most

$$P(\nu;l,1-\epsilon) \;=\; \sum_{i<\nu} \binom{l}{i} \epsilon^{l-i} (1-\epsilon)^i, \tag{4}$$

and this bound is sharp for large T.

The probability (4) has been well studied; see [5] for a survey. Whenever

$$a := \nu / l > 1-\epsilon,$$

Chernoff's inequality [5, Theorem 1] gives us

$$P(\nu;l,1-\epsilon) \leqslant \exp\left(-l\left(a\log\left(\frac{a}{1-\epsilon}\right) + (1-a)\log\left(\frac{1-a}{\epsilon}\right)\right)\right).$$

Let $\eta<1$ be a positive real parameter. In order to ensure $P(\nu;l,1-\epsilon)<\eta$ it suffices to have

$$l\left(a\log\left(\frac{a}{1-\epsilon}\right) + (1-a)\log\left(\frac{1-a}{\epsilon}\right)\right) > \log(1/\eta).$$

Now thanks to [40, Lemma 2(a)] we have

$$a\log\left(\frac{a}{1-\epsilon}\right) + (1-a)\log\left(\frac{1-a}{\epsilon}\right) \geqslant 2\,(a-(1-\epsilon))^2,$$

so it suffices to ensure that

$$(\nu - l\,(1-\epsilon))^2 > \frac{\log(1/\eta)}{2}\, l. \tag{5}$$

Now let us take $\nu = c\,(1-\epsilon)\,l$ with $c>1$, after which (5) is equivalent to

$$(c-1)^2 > \frac{\log(1/\eta)}{2\,l\,(1-\epsilon)^2}. \tag{6}$$

For fixed η and large l (i.e. for large D), it follows that we may take c arbitrarily close to 1.

Summarizing, we may thus take $l \approx \nu/(1-\epsilon)$ in order to recover an average of at least $(1-\eta)\,t$ correct terms, where η can be taken arbitrarily close to 0:

PROPOSITION 1. *Assume* $\mathbf{H_{red}}$ *and let* τ, ϵ, l, $r_1,\ldots,r_l$, *and* ν *be parameters as above. Given* $\eta<1$, *assume that* $\nu = c\,(1-\epsilon)\,l$, *where* c *satisfies* (6). *Then Algorithm 3 returns at least* $(1-\eta)\,t$ *correct terms of* f *on average.*

4.5. Probabilistic complexity analysis

Let us now analyze the running time of Algorithm 3. Taking $l \approx \nu/(1-\epsilon)$, the cost of step 1 is proportional to

$$\frac{\tau}{1-\epsilon} \;=\; \tau\,\mathrm{e}^{1/\tau},$$

and $\tau\,\mathrm{e}^{1/\tau}$ reaches its minimum value e at $\tau=1$. This means that the total complexity is best when τ is close to 1. In other words, this prompts us to take $\tau=1$, $r_1 \approx \cdots \approx r_l \approx T$, $\nu \approx \log D/\log T$, and $l \approx \nu/(1-\mathrm{e}^{-1})$. For this choice of parameters, we obtain the following heuristic complexity bound:

PROPOSITION 2. *Assume that* $\boldsymbol{H_{red}}$ *and* $\log D = O(T)$. *Given* $0<\eta<1$ *and a diversified polynomial* f *of degree* $d \leqslant D$ *and with* $t \leqslant T$ *terms, there exists a Monte Carlo probabilistic algorithm which computes at least* $(1-\eta)\,t$ *terms of* f *in time*

$$O^{\flat}(L\,T\log D\log q).$$

Proof. We take r_i to be the i-th smallest prime numbers that is larger than T, so that $\nu = O(\log D / \log T)$ is the smallest number with $r_1 \cdots r_\nu > D$. We also take $\tau = 1$, $\epsilon = 1 - 1/\mathrm{e}$, and let $l = O(\log D / \log T)$ be smallest such that (6) is satisfied for $c = \nu / (l(1-\epsilon))$. Combining [6] and [1], we may compute $r_1, \ldots, r_l$ in time $O^\flat(T \log D)$.

Now the running time of step 1 of Algorithm 3 is

$$O((\mathsf{M}_q(r_1) + \cdots + \mathsf{M}_q(r_l))\, L).$$

With $l = O(\log D / \log T)$, this cost simplifies to

$$O^\flat(L T \log(qT) \log D / \log T) = O^\flat(L T \log q \log D).$$

Step 3 may be obtained by sorting the coefficients of the $f^{[r_k]}$, in time

$$O(l T \log T \log q) = O(T \log D \log q).$$

Using fast Chinese remaindering, step 4 takes time $T\tilde{O}(\log \nu \log D) = O^\flat(T \log D)$. □

Remark. If $q \gg T^2$, then $f(\alpha x)$ is diversified with high probability for a random choice of $\alpha \in \mathbb{F}_q \setminus \{0\}$: see [16]. In the range where $q = \Theta(T)$ and $q = O(T^2)$, it is possible to work with a slightly weaker assumption: we say that f is *weakly diversified* if $\{c_1, \ldots, c_t\}$ is of size $\Theta(t)$. If $q = \Theta(T)$, then the polynomial $f(\alpha x)$ is still weakly diversified, for a random choice of $\alpha \in \mathbb{F}_q \setminus \{0\}$. If f is only weakly diversified and $t' := \{1 \leqslant i \leqslant t : \forall j \neq i, c_i \neq c_j\}$, then our analysis can be adapted to show that Algorithm 3 returns about $(1-\eta)\, t'$ correct terms of f on average. Finally, in the range where $q = o(T)$, we need to work over a field extension $\mathbb{F}_{q^s}$ with $q^s \geqslant T$, which leads to an additional arithmetic overhead of $O^\flat(\log T / \log q)$.

Remark. Let us show that with high probability, the polynomial f^* returned by Algorithm 3 only contains correct terms of f (although it does not necessarily contain all terms). For this, we make the additional hypothesis that the coefficients of f are essentially random non-zero values in $\mathbb{F}_q$ (which is typically the case after a change of variables $f(x) \rightsquigarrow f(\alpha x)$, where $\alpha \in \mathbb{F}_q \setminus \{0\}$ is random).

Now assume that some coefficient c in step 4 gives rise to a term $c x^e$ that is not in f. Then for every $k \in \mathcal{K}$, there should be at least two terms in f that collide modulo r_k and for which the sum of the corresponding coefficients equals c. The probability that this happens for a fixed $k \in \mathcal{K}$ is bounded by $(q-1)^{-1}$ and the probability that this happens for all $k \in \mathcal{K}$ is bounded by $(q-1)^{-\nu'}$, where $\nu' \approx \nu$ is minimal with $r_{l-\nu'+1} \cdots r_l > D$.

4.6. Estimating the number of terms t

For the algorithms in this section, we assumed that a bound T for t was given. It turns out that a variant of our probabilistic analysis can also be used for the efficient computation of a rough estimate for t. This yields an interesting alternative to the naive doubling strategy described in section 3.1.

Let us still assume that $\mathbf{H_{red}}$ holds. We will also assume that colliding terms rarely cancel out (which holds with high probability if q is sufficiently large). This time, we compute $f(x)$ rem $(x^B - 1)$ for $B := \alpha t$, where $\alpha < 1$ is to be determined, and let N be the number of terms in this remainder. When randomly distributing t balls over B boxes, the probability that none of the balls lands in a particular box is $(1 - 1/B)^t$. Consequently, the expected number of boxes with no ball is $(1-1/B)^t B$, whence

$$B - N \approx \left(1 - \frac{1}{B}\right)^t B \approx \mathrm{e}^{-\frac{1}{\alpha}} B.$$

It follows that

$$\frac{1}{\alpha} \approx \log\left(\frac{B}{B-N}\right),$$

and thus

$$t = \frac{B}{\alpha} \approx B \log\left(\frac{B}{B-N}\right). \tag{7}$$

By doubling B until $B - N \gtrsim \sqrt{B}$, we may then use the approximation (7) as a good candidate for t. Notice that we have $B - N \approx \sqrt{B}$ when $B \approx 2t/\log t$.

4.7. Conclusion

Cyclic extension methods for sparse interpolation are attractive due to their generality and the possibility to derive deterministic complexity bounds. However, even their most efficient probabilistic versions suffer from the overhead of arithmetic operations in cyclic extension algebras $K[x]/(x^r - 1)$.

The matching strategy based on diversification leads to the best practical complexity bounds, as shown in section 4.5. Assuming $\mathbf{H_{red}}$, $\log D = O(T)$, and $q = \Theta(T)$, we have given a Monte Carlo algorithm for sparse interpolation of complexity $O^\flat(LT \log D \log T)$. The case when $q = o(T)$ can be reduced to this case using a field extension of degree $O(\log T / \log q)$. Assuming only $\mathbf{H_{red}}$ and $\log D = O(T)$, we thus obtain a probabilistic algorithm that runs in time

$$O^\flat(LT \log D \log (qT)). \tag{8}$$

5. UNIVARIATE INTERPOLATION USING GEOMETRIC PROGRESSIONS

Prony's method is one of the oldest and most celebrated algorithms for sparse interpolation of univariate polynomials. It is based on the evaluation of f at points in a geometric progression. Since there are many variants, we keep our presentation as general as possible. As in the previous section, assume that

$$f = c_1 x^{e_1} + \cdots + c_t x^{e_t} \tag{9}$$

and that we know bounds D and T for the degree and the number of terms of f.

Algorithm 4
Input: a black box polynomial $f(x)$, a degree bound D, a sparsity bound T
Output: the sparse interpolation f^* of f

1. Find a suitable element $\omega \in K$ with multiplicative order $r \geqslant \max(D, 2T)$, while replacing K by an extension if necessary.
2. Evaluate $f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{2T-1})$.
3. Compute a minimal $t \leqslant T$, a monic $\Lambda \in K[z]$ of degree t, and an $N \in K[z]$ of degree $< t$, such that the following identity holds modulo $O(z^{2T})$:

$$\sum_{k \in \mathbb{N}} f(\omega^k) z^k = \sum_{1 \leqslant i \leqslant t} \frac{c_i}{1 - \omega^{e_i} z} = \frac{N(z)}{\Lambda(z)}. \tag{10}$$

4. Find the roots ω^{-e_i} of $\Lambda := z^t + \Lambda_{t-1} z^{t-1} + \cdots + \Lambda_0 \in K[z]$.
5. Compute the discrete logarithms of the roots to base ω to discover the exponents $e_1, \ldots, e_t$ of f as in (9).
6. Compute $c_{e_1}, \ldots, c_{e_t}$ from $f(1), \ldots, f(\omega^{t-1})$ and $e_1, \ldots, e_t$, using linear algebra.

It is well known that steps 3 and 6 can be performed in time $O(\mathsf{M}_K(T)\log T)$, through fast Padé approximation [10, 38] in the case of step 3, and using a transposed version of fast multipoint interpolation [9, 11] for step 6. If $K=\mathbb{F}_q$, then this bound becomes $O^\flat(T(\log T)^2\log q)$. The efficiency of steps 4 and 5 highly depends on the coefficient field K. In the remainder of this section, we will discuss this issue in detail in the case when $K=\mathbb{F}_q$ is a finite field.

5.1. Root finding

Finding the roots of a univariate polynomial over a finite field is a well-known and highly studied problem in computer algebra. The most efficient general purpose algorithm for this task is due to Cantor–Zassenhaus [12]. It is probabilistic and computes the roots of Λ in time $O^\flat(T(\log T)^2(\log q)^2)$.

In [17, 18], several alternative methods were designed for the case when $r=r_1r_2$ with $r_1=O(T)$ and $r_2\leqslant T^{O(1)}$ smooth (in the sense that $\pi=O^\flat(1)$ for each prime factor π of r_2). The idea is to proceed in three steps:

1. We first compute the r_2-th Graeffe transform $G_{r_2}(\Omega)$ of Ω, whose roots are the r_2-th powers of the roots of Ω. This step can be done in time $O^\flat(T(\log T)^2\log q)$ by [17, Proposition 5].
2. We next compute the roots of $G_{r_2}(\Omega)$ through an exhaustive evaluation at all r_1-th roots of unity. This step takes time $O^\flat(T\log T\log q)$.
3. We finally lift these roots back up to the roots of Ω. This can be done in time $O^\flat(T(\log T)^2\log q\log r_2)=O^\flat(T(\log T)^3\log q)$, using g.c.d. computations.

Altogether, this yields a sparse interpolation method of cost $O^\flat(T(\log T)^3\log q)$.

The backlifting of single roots can be accelerated using so-called "tangent Graeffe transforms". The idea is to work over the ring $R:=K[\epsilon]/(\epsilon^2)$ instead of K. Then $\alpha\in K$ is a root of a polynomial $P(z)\in K[z]$ if and only if $\alpha+\epsilon\in R$ is a root of the polynomial $P(z-\epsilon)\in R[z]$. Now if we know a single root $(\alpha+\epsilon)^{r_2}=\alpha^{r_2}+r_2\alpha^{r_2-1}\epsilon$ of $G_{r_2}(\Omega(z-\epsilon))$, then we may retrieve α using one division of α^{r_2} by $r_2\alpha^{r_2-1}$ and one multiplication by r_2 (note that r_2 is invertible in $\mathbb{F}_q$ since r_2 divides $q-1$). In other words, the backlifting step can be done in time $O^\flat(T\log q)$, using $O(T)$ operations in $\mathbb{F}_q$.

However, this method only works for single roots α. When replacing $\Omega(z)$ and $\Omega(z-\lambda)$ for a randomly chosen $\lambda\in\mathbb{F}_q$, the polynomial $G_{r_2}(\Omega(z-\lambda))$ can be forced to admit a non-trivial proportion of single roots with high probability. However, these roots are no longer powers of ω, unless we took $r=q-1$. Assuming that $r=q-1$ and using several shifts λ, it can be shown [17, Proposition 12] that the tangent Graeffe method yields a sparse interpolation method of complexity $O^\flat(T\log T(\log q)^2)$.

5.2. Discrete logarithms

The discrete logarithm problem in abelian groups is a well-known problem in computational number theory. If r is smooth, then Pohlig–Hellman's algorithm provides an efficient solution; it allows step 5 of Algorithm 4 to be performed in time $O^\flat(T\log r\log q)$. Under the assumption that we may take $r=T^{O(1)}$, this bound reduces to $O^\flat(T\log T\log q)$.

Again, the same bound still holds if $r=r_1r_2$ with $r_1=O(T)$ and r_2 smooth. Indeed, in that case, we may tabulate the powers $\omega^0,\omega^{r_2},\omega^{2r_2},\ldots,\omega^{(r_1-1)r_2}$. This allows us to efficiently determine the discrete logarithms of $\omega^{e_1r_2},\ldots,\omega^{e_tr_2}$ with respect to ω^{r_2}, which yields the exponents $e_1,\ldots,e_t$ modulo r_1. We next use Pohlig–Hellman's algorithm to compute $e_1,\ldots,e_t$.

5.3. Field extensions

If $\max(T, 2D)$ exceeds q (or if $q-1$ admits no suitable factors r that allows us to apply the above methods), then we may need to work over an extension field $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$. Notice that this requires our black box representation of f to accept inputs in such extension fields.

Since evaluations over $\mathbb{F}_{q^s}$ are at least s times more expensive, it is important to keep s as small as possible. If $T > q$, then we must necessarily have $q^s \geqslant 2T+1$, whence $s \geqslant \log(2T+1)/\log q$. In general, we want to take $s = O(\lceil \log T / \log q \rceil)$. Since we also need $r \geqslant D$ in step 1, this leads to the constraint $D = T^{O(1)}$. Under this hypothesis and using the observations from section 2, it is likely that a suitable extension order s and divisor $r \mid (q^s - 1)$ can indeed be found.

Denoting by $\mathsf{M}_q(s)$ the cost of multiplication of polynomials of degree s over $\mathbb{F}_q$, the total cost of sparse interpolation then becomes

$$O^\flat((L + (\log T)^3)\, T\, \mathsf{M}_q(s)). \tag{11}$$

5.4. Exploiting the Frobenius map

An interesting question is whether we can reduce the number of evaluation points when working over an extension field $\mathbb{F}_{q^s}$. Indeed, if ϕ is the Frobenius map of $\mathbb{F}_{q^s}$ over $\mathbb{F}_q$, then $f(\phi(a)) = \phi(f(a))$ for all $a \in \mathbb{F}_{q^s}$. If $\mathbb{F}_{q^s} = \mathbb{F}_q[a]$, then this allows us to obtain the evaluations at the s distinct points $a, \phi(a), \ldots, \phi^{s-1}(a)$ using a single evaluation at a. In step 2 of Algorithm 4, we can therefore avoid the evaluations at ω^i for $i = q, 2q, 3q, \ldots$ and gain a constant factor $q/(q-1)$ for large s. Similarly, we may compute all values $f(1), f(\omega), \ldots, f(\omega^{r-1})$ using approximately r/s evaluations of f only; whenever $r/(2T)$ is small, this allows us to gain a factor $2Ts/r$. It would be interesting to know whether it is possible to do better and regain a factor $\Theta(s)$ in general.

5.5. Traces

Besides working in an extension field, it may also be interesting to perform part of the computations over a subfield of $\mathbb{F}_q$. Indeed, the main aim of steps 4 and 5 is to find the exponents of f. Now assume that $\mathbb{F}_q$ admits a subfield $\mathbb{F}_{q'}$ with $\max(2T-1, D) < q'$ and let $\mathrm{Tr}: \mathbb{F}_q \to \mathbb{F}_{q'}$ be the corresponding trace function. Then f and $\mathrm{Tr}\, f$ are likely to admit approximately the same exponents. Taking $\omega \in \mathbb{F}_{q'}$ in step 1, we may thus replace $f(1)$,..., $f(\omega^{2T-1})$ by their traces after step 2, and determine the exponents of $\mathrm{Tr}\, f$ instead of f. Although this does not allow us to speed up steps 2 and 6, we do gain a factor of at least $\log q / \log q'$ in steps 4 and 5.

5.6. Combining interpolations for several moduli r

Once the order r has been fixed, Algorithm 4 essentially allows us to interpolate $f(x)$ modulo $x^r - 1$. With respect to the cyclic extension approach, the main difference is that one expensive evaluation of f over the extension ring $K[x]/(x^r-1)$ is replaced by $2T$ cheap evaluations over K plus $O^\flat(T(\log T)^2)$ scalar operations.

If $D \gg T$, then we may also evaluate $f(x)$ modulo $x^r - 1$ for different moduli $r = r_1, \ldots, r_l$ and recombine the results using one of matching strategies from section 4. However, in the present context, we are not completely free to pick our moduli, since we need corresponding primitive roots of unity $\omega_1 \in \mathbb{F}_{q^{s_1}}, \ldots, \omega_l \in \mathbb{F}_{q^{s_l}}$ of orders $r_1, \ldots, r_k$ in small extensions $\mathbb{F}_{q^{s_1}}, \ldots, \mathbb{F}_{q^{s_l}}$ of $\mathbb{F}_q$.

Let us focus more specifically on Algorithm 3, which requires in particular that $q \geqslant T$. We need $r_1, \ldots, r_l > 2T$ to be suitable for steps 4 and 5, so $r_k = r_{k,1} r_{k,2}$ with $r_{k,1} = O(T)$ and $r_{k,2}$ is smooth. On the other hand, we may relax the conditions on $r_1, \ldots, r_l$. In this case, the complexity does not depend on τ, so it is better to choose the r_k much larger than T, preferably of the order of T^2. It is also not necessary that $r_1, \ldots, r_l$ be pairwise coprime: it suffices that $\operatorname{lcm}(r_{k_1}, \ldots, r_{k_\nu}) > D$ for any $1 \leqslant k_1 < \cdots < k_\nu \leqslant l$. Ideally speaking, we should have $l \approx \nu \approx \log D / \log T$.

Although there is no *a priori* reason for suitable $r_1, \ldots, r_l$ of this kind to exist, we expect that this will often be the case in practice, as long as $\log D = O(\log T)$. Evidence in this direction will be presented in sections 6.3 and 6.4 below, under even stronger constraints on the orders $r_1, \ldots, r_l$. Assuming that we are indeed able to find suitable $r_1, \ldots, r_l$, the overall runtime complexity becomes

$$O^\flat((L + (\log T)^3)\, T\, (\mathsf{M}_q(s_1) + \cdots + \mathsf{M}_q(s_l))).$$

When using naive arithmetic with $\mathsf{M}_q(s) = O^\flat(s^2 \log q)$ and assuming that $s_i = O(i)$, this complexity bound simplifies into

$$O^\flat\left((L + (\log T)^3)\, T \left(\frac{\log D}{\log T}\right)^3 \log q\right).$$

5.7. Conclusion

In summary, the efficiency of the geometric progression approach over a finite field rests on our ability to find suitable divisors of $q^s - 1$ for small values of s. If $D \leqslant T^2$ and $T \leqslant q$, then we essentially need an order $r \mid (q^s - 1)$ of the type $r = r_1 r_2$ with $r_1 \approx T$, r_2 smooth, and s small. By what has been said in section 2, it is very likely that such r and s always exist. If $D = T^{O(1)}$ and $T \leqslant q$, then it remains likely that various divisors of this type can be combined, as explained in section 5.6. If $q < T$, then we first need to replace K by a suitable extension.

Assuming that $D = T^{O(1)}$ and that suitable orders as above can indeed be found, the cost of the geometric progression approach is bounded by

$$O^\flat\left((L + (\log T)^3)\, T \left(\frac{\log D}{\log T}\right)^3 \log (q T)\right). \tag{12}$$

In favorable cases when $D < q = T^{O(1)}$ and $q - 1$ is smooth, we obtain the complexity bound

$$O^\flat((L + \log T \log q)\, T \log q), \tag{13}$$

instead of (12), by using the tangent Graeffe method.

In comparison with algorithms based on cyclic extensions, the main advantage of the algorithms in this section is that we avoid expensive evaluations over cyclic extension rings. On the other hand, the cost of the root finding step may dominate the cost of the evaluations of f whenever $L = o((\log T)^3)$; in that case, cyclic extension methods may become competitive. Methods based on geometric progressions are also less suited for the supersparse case.

Remark 3. For practical implementations, it is not convenient to use an *a priori* comparison in order to determine which of the L or $(\log T)^3$ terms dominates the cost in (12). Since we usually try to interpolate f for increasingly large bounds T, it is better to test whether step 1 of Algorithm 4 requires more running time than step 3 as we increase T. Whenever the cost of step 3 starts to dominate, we may switch to a cyclic extension style approach (or an FFT based approach, to be presented next).

6. UNIVARIATE SPARSE INTERPOLATION USING FFTS

Geometric progression style algorithms for sparse interpolation admit the big advantage that they have a sharp complexity with respect to L. However, if evaluations of f are cheap, then the $(\log T)^3$ term in (11) may dominate L. In this section, we will investigate a strategy to reduce the dependency in $\log T$, at the price of $O(1)$ more evaluations. The idea is to more aggressively exploit the observations from sections 5.6 and 5.4. Alternatively, we can regard our proposal as a careful adaptation of the cyclic extension approach that allows us to replace evaluations over cyclic extensions by evaluations over K itself.

6.1. Fast evaluation modulo $x^r - 1$

For a fixed parameter τ that we control and a modulus r close to τT, let us first study how to evaluate $f(x)$ efficiently modulo $x^r - 1$. Instead of evaluating f at only $2T-1$ points $1, \omega, \ldots, \omega^{2T-1}$ for a primitive r-th root of unity ω (as in step 2 of Algorithm 4), we rather evaluate f at all r-th roots of unity $1, \omega, \ldots, \omega^{r-1}$. The advantage is that we may then use FFT-based techniques as a replacement for the remaining steps of Algorithm 4.

Moreover, if $K = \mathbb{F}_q$ and ω lives in an extension $\hat{K} = \mathbb{F}_{q^s}$ of K, then it suffices to evaluate f at only $\sim r/s$ points in order to determine all values $f(1), \ldots, f(\omega^{r-1})$ using the Frobenius map. Recovering $f(x)$ modulo $x^r - 1$ from these values can also be done efficiently using the inverse Frobenius FFT [23].

Algorithm 5
Input: a black box polynomial $f(x)$ over $\mathbb{F}_q$ and $\omega \in \mathbb{F}_{q^s}$ of order $r \in \mathbb{N}$
Output: a polynomial $f^* \in \mathbb{F}_q[x]$ of degree $< r$ with $f = f^*$ modulo $x^r - 1$

1. Let Ω be a section of $\{1, \omega, \ldots, \omega^{r-1}\}$ under the Frobenius map ϕ that is suitable for the Frobenius DFT.
2. Compute $f(\xi)$ for each $\xi \in \Omega$.
3. Compute the coefficients of f^* by applying one inverse Frobenius DFT of order r to $(f(\xi))_{\xi \in \Omega}$ and return f^*.

Assuming for simplicity that $|\Omega| = \Theta(r/s)$ and that the computation of an inverse Frobenius DFT is $\Theta(s)$ times faster than the computation of a full DFT, we note that the cost of one run of Algorithm 5 is bounded by

$$O^{\flat}\left((L + \log T)\, T\, \tau\, \frac{\mathsf{M}_q(s)}{s} \right). \tag{14}$$

6.2. Recombination into approximate sparse interpolations

If $D \gg T$ and $q \geqslant T$, then we wish to apply a similar strategy as in section 5.6 and recombine the values of $f(x)$ rem $(x^r - 1)$ for various moduli $r = r_1, \ldots, r_l$. This time, in order to perform step 1 of Algorithm 3 using Algorithm 5, we need the orders $r_1, \ldots, r_l$ to be close to T. With $\tau \approx 1$ as in section 4.4, we thus assume $r_1 = \tau_1 T, \ldots, r_l = \tau_l T$ with $\tau_1, \ldots, \tau_l \approx \tau \approx 1$. As in section 5.6, we also impose the condition that $\operatorname{lcm}(r_{k_1}, \ldots, r_{k_\nu}) > D$ for any $1 \leqslant k_1 < \cdots < k_\nu \leqslant l$. Ideally speaking, we have $\nu \approx \log D / \log T$ and $l \approx \nu / (1 - \mathrm{e}^{-\tau})$.

Under these conditions, thanks to (14), the overall running time of Algorithm 3 is bounded by

$$O^{\flat}\left((L+\log T)\,T\left(\tau_1\frac{\mathsf{M}_q(s_1)}{s_1}+\cdots+\tau_l\frac{\mathsf{M}_q(s_l)}{s_l}\right)\right). \tag{15}$$

When using naive arithmetic with $\mathsf{M}_q(s)=O^{\flat}(s^2\log q)$ and assuming that $s_i=O(i)$, this bound simplifies into

$$O^{\flat}\left((L+\log T)\,T\left(\frac{\log D}{\log T}\right)^2\log q\right).$$

Due to our hypothesis that $l\approx\nu/(1-\mathrm{e}^{-\tau})$ and the analysis from section 4.4, we can still expect the algorithm to return about half of the terms of f.

Remark 4. If $q<T$, then we need to replace K by an extension of degree at least $\deg T/\deg q$ before being able to diversify f, without being able to profit from the Frobenius map. This leads to an incompressible overhead of $\deg T/\deg q$. Nevertheless, in the special case when the exponents $e_1,\ldots,e_t$ are already known, the matching step 4 of Algorithm 3 can be simplified, and working over an extension can be avoided.

6.3. Example over $\mathbb{F}_2$

In order to apply Algorithm 3 as in the previous section, we need primitive roots of unity of suitable orders $r_1,\ldots,r_l$ in algebraic extensions of $\mathbb{F}_q$. Although we have no general theory, it is instructive to study a few practical examples in order to gain insight what we may reasonably hope for. In this section, we consider the case when $q=2$, $T=10^6$, and $D=10^{18}$.

First of all, we need to replace $\mathbb{F}_q$ by a sufficiently large extension $\mathbb{F}_{q'}$ in order to diversify f (at least in the weak sense from the end of section 4.2.3). Since it is favorable to take

$$q'>\log 10^6/\log 2\approx 20$$

as smooth as possible, we opt for $q'=30$. For $k=1,2,\ldots$, we next take our orders $r_k\approx 10^6$ with $r_k\mid(q^{30s_k}-1)$ and s_k as small as possible, and such that

$$\Lambda_k:=\operatorname{lcm}(r_1,\ldots,r_k)$$

is as large as possible:

$$\begin{array}{rcrcrcl}
s_1 &=& 1 & r_1 &=& 1549411 &=& 31\cdot 151\cdot 331 & \Lambda_1 &\approx& 1.5\cdot 10^6\\
s_2 &=& 2 & r_2 &=& 1047553 &=& 13\cdot 61\cdot 1321 & \Lambda_2 &\approx& 1.6\cdot 10^{12}\\
s_3 &=& 3 & r_3 &=& 1701703 &=& 73\cdot 23311 & \Lambda_3 &\approx& 2.8\cdot 10^{18}\\
s_4 &=& 3 & r_4 &=& 1186911 &=& 3^2\cdot 11\cdot 19\cdot 631 & \Lambda_4 &\approx& 3.2\cdot 10^{24}\\
s_5 &=& 4 & r_5 &=& 1048577 &=& 17\cdot 61681 & \Lambda_5 &\approx& 3.4\cdot 10^{30}\\
s_6 &=& 4 & r_6 &=& 1729175 &=& 5^2\cdot 7\cdot 41\cdot 241 & \Lambda_6 &\approx& 5.9\cdot 10^{36}\\
s_7 &=& 5 & r_7 &=& 1016801 &=& 251\cdot 4051 & \Lambda_7 &\approx& 6.0\cdot 10^{42}\\
s_8 &=& 5 & r_8 &=& 1082401 &=& 601\cdot 1801 & \Lambda_8 &\approx& 6.5\cdot 10^{48}\\
s_9 &=& 5 & r_9 &=& 1108811 &=& 11\cdot 100801 & \Lambda_9 &\approx& 6.6\cdot 10^{53}\\
s_{10} &=& 6 & r_{10} &=& 1134021 &=& 3\cdot 7\cdot 54001 & \Lambda_{10} &\approx& 3.6\cdot 10^{58}
\end{array}$$

Taking $\tau=1$, we obtain $\nu\approx 18/6=3$ and $l\approx\nu\mathrm{e}$. To be on the safe side, we take $l=10$. The minimum least common multiple of three (resp. four) orders among $r_1,\ldots,r_{10}$ is

$$\operatorname{lcm}(r_4,r_9,r_{10})\approx 4.5\cdot 10^{16}$$

(resp. $\operatorname{lcm}(r_4, r_6, r_9, r_{10}) \approx 1.1 \cdot 10^{22}$), so we have $\nu = 4$. Notice that some of the $\tau_k = r_k / T$ are quite a bit larger than τ, with an average of $\tau' = (\tau_1 + \cdots + \tau_{10}) / 10 \approx 1.26$. For $l = 10$ and $\mathrm{e}^{-1/1.26} l \approx 4.52 > \nu$, we therefore consider it likely that the condition $\operatorname{lcm}(r_k : k \in \mathcal{K}) > D$ in step 4 of Algorithm 3 is satisfied with probability $> 1/2$ (a rigorous analysis would be welcome).

6.4. Example over $\mathbb{F}_{1299743}$

As our second example, consider the case when $q = 1299743$ (as in section 2), $T = 10^6$, and $D = 10^{18}$. This time $\mathbb{F}_q$ contains at least T elements, so we may directly work over $\mathbb{F}_q$. Proceeding as in the previous subsection, we may now take:

$$\begin{array}{lll}
s_1 = 1 & r_1 = 1299742 = 2 \cdot 649871 & \Lambda_1 \approx 1.3 \cdot 10^{6} \\
s_2 = 2 & r_2 = 1299744 = 2^5 \cdot 3^2 \cdot 4513 & \Lambda_2 \approx 8.4 \cdot 10^{11} \\
s_3 = 4 & r_3 = 1006325 = 5^2 \cdot 40253 & \Lambda_3 \approx 8.5 \cdot 10^{17} \\
s_4 = 4 & r_4 = 1678714 = 2 \cdot 193 \cdot 4349 & \Lambda_4 \approx 7.1 \cdot 10^{23} \\
s_5 = 5 & r_5 = 1690111 = 701 \cdot 2411 & \Lambda_5 \approx 1.2 \cdot 10^{30} \\
s_6 = 6 & r_6 = 1119937 = 7 \cdot 13 \cdot 31 \cdot 397 & \Lambda_6 \approx 1.4 \cdot 10^{36} \\
s_7 = 8 & r_7 = 1196324 = 2^2 \cdot 17 \cdot 73 \cdot 241 & \Lambda_7 \approx 4.0 \cdot 10^{41} \\
s_8 = 9 & r_8 = 1185702 = 2 \cdot 3 \cdot 7^2 \cdot 37 \cdot 109 & \Lambda_8 \approx 1.1 \cdot 10^{46} \\
s_9 = 10 & r_9 = 1376122 = 2 \cdot 11 \cdot 71 \cdot 881 & \Lambda_9 \approx 7.8 \cdot 10^{51} \\
s_{10} = 11 & r_{10} = 3423619 = 23 \cdot 148853 & \Lambda_{10} \approx 2.7 \cdot 10^{58}
\end{array}$$

With $\tau = 1$, $l = 10$, and $\nu = 4$ as before, the minimum least common multiple of ν orders among $r_1, \ldots, r_{10}$ is $\operatorname{lcm}(r_2, r_6, r_7, r_8) \approx 1.2 \cdot 10^{22}$. Again, the "effective" mean value $\tau' = (\tau_1 + \cdots + \tau_{10}) / 10 \approx 1.5$ satisfies $\mathrm{e}^{-1/\tau'} l \approx 5 > \nu$.

Notice that s_k approximately grows like $s_k \approx 1.1 \cdot k$. On the first example from the previous subsection, we rather have $s_k \approx 0.7 \cdot k$. This suggests that the growth of s_k might often be somewhat above $(\log T / \log q)\, k$.

As an optimization, we also notice that it should be better to take $\tau_k = r_k / T$ somewhat larger for small values of s, so as to balance the terms $\tau_k \mathsf{M}_q(s_k) / s_k$ in the sum (15). For instance, the last number r_{10} is quite a lot larger than T; it should be more efficient to resort to a larger extension degree $s_{10} = 13$ and take $r_{10} = 1220444 = 2^2 \cdot 305111$. Another option would be to include $s'_3 = 3$ and $r'_3 = 6680137$. In addition, by increasing r_3 and r_4, it might also be possible to take $l = 9$.

6.5. Sparse interpolation over the rationals

As mentioned in the introduction, one important application of sparse interpolation over finite fields is sparse interpolation over the rational numbers. In that case, we typically combine sparse interpolation over different prime fields $\mathbb{F}_q$ for which $q - 1$ is divisible by a high power of two.

We usually proceed in two stages. We first determine the exponents $e_1, \ldots, e_t$ of f (typically using a few primes q only). Once the exponents are known, the computation of the coefficients modulo further primes q (typically many ones) becomes more efficient. We finally use the Chinese remainder theorem and rational number reconstruction to determine the actual coefficients.

The techniques from this section may help to accelerate both stages. If f is normally sparse, then we may directly apply the algorithms from above to find the exponents. If f is supersparse, then we will now present a multimodular variant of the strategy from section 3.2. Given a number $r \geqslant T$, the aim is to determine with high probability those exponents e_i such that $e_i \neq e_j$ for all $j \neq i$ and use this to reconstruct an approximate sparse interpolation of f. We next apply Algorithm 1.

So let $r \geqslant T$ be fixed and consider successive prime numbers $q_1, q_2, \ldots$ for which $q_k - 1$ is divisible by r. Let l be minimal such that $q_1 \cdots q_l > D$. We first compute $f^{[q_k,r]} := (f \bmod q_k) \operatorname{rem} (x^r - 1)$ for $k = 1, \ldots, l$ using Algorithm 5. We next reconstruct $(f \bmod q_1 \cdots q_l) \operatorname{rem} (x^r - 1)$ using Chinese remaindering. In a similar way, we compute $(xf' \bmod q_1 \cdots q_l) \operatorname{rem} (x^r - 1)$. For any coefficient $c_i x^{e_i}$ of f such that $c_j \neq c_i$ modulo r for all $j \neq i$, we thus obtain both coefficients c_i and $c_i e_i$ modulo $q_1 \cdots q_l$. Whenever $\gcd(q_1 \cdots q_l, c_i) = 1$, this allows us to retrieve e_i. By considering a few more primes $q_{l+1}, \ldots, q_{l+\delta}$ for sanity checking, we may thus compute e_i with high probability. The running time of this algorithm is bounded by $O^\flat((L + \log T)\, T \log D)$.

If the exponents $e_1, \ldots, e_t$ are known, then we may also use the techniques from this section to further speed up the computation of the coefficients $f_{e_1}, \ldots, f_{e_t}$. For this, let $t \leqslant r_1 \leqslant r_2 \leqslant \cdots$ be an increasing sequence of pairwise coprime numbers. Assume that the numerators and denominators of the coefficients f_{e_i} are bounded by B with $\log B = o(t)$, and assume that $r_k = O(t)$ for the first $O(\log B)$ terms of the sequence (r_k). For each k, let q_k be the smallest prime number such that r_k divides $q_k - 1$.

Starting with $\mathcal{K}_1 = \cdots = \mathcal{K}_t = \emptyset$, we now repeat the following loop for $k = 1, 2, \ldots$ until $\prod_{k \in \mathcal{K}_i} q_k > B^2$ for all $i = 1, \ldots, t$:

- We compute $(f \bmod q_k) \operatorname{rem} (x^{r_k} - 1)$ using Algorithm 5.
- For $i = 1, \ldots, t$, if $e_i \neq e_j$ modulo r_k for all $j \neq i$, then set $\mathcal{K}_i := \mathcal{K}_i \cup \{k\}$.

At the end of this loop, for $i = 1, \ldots, t$, we may recover the coefficient f_{e_i} from the coefficients of $x^{e_i \operatorname{rem} r_k}$ in $(f \bmod q_k) \operatorname{rem} (x^{r_k} - 1)$, where k runs over $\mathcal{K}_i$. With high probability, this algorithm runs in time $O^\flat((L + \log t)\, t \log B)$, modulo the heuristic $\mathbf{H_{red}}$.

6.6. Further remarks

The FFT based technique from this section is intended to be used in combination with a probabilistic recombination method such as Algorithm 3 or the technique from the previous subsection. In the most favorable case when we are always able to pick r close to T, we have seen that approximately T/e terms of f are expected not collide modulo $x^r - 1$. This means that we have evaluate f at least $\mathrm{e}\, T$ times in order to determine its sparse interpolation. Recall that only $2T - 1$ evaluations were required with Algorithm 4. An interesting question is whether there exist any compromises or improvements.

One idea is to exploit colliding terms better. For instance, assume that we are computing $f^{[r_k]} := f \bmod (x^{r_k} - 1)$ for various moduli r_k and that $f^{[r_1]}$ contains a term that comes from the sum of exactly two terms $c_i x^{e_i}$ and $c_j x^{e_j}$ with $e_i \equiv e_j$ modulo r_1. Now suppose that we are able to determine $c_i x^{e_i}$ from the reductions $f^{[r_k]}$ for a few other moduli $r_k \neq r_1$. Then we might reuse $f^{[r_1]}$ to compute the term $c_j x^{e_j}$ using a simple subtraction.

Another idea is to optimize the (non-tangent) Graeffe transform variant of the geometric progression method. With the notations from section 5.1, we have seen how to achieve a complexity of $O^\flat((L + (\log T)^2 \log r_2) \log q)$. When taking $r_2 = T^{o(1)}$, this complexity bound reduces to $O^\flat((L + (\log T)^2) \log q)$, whereas only about $O(t/r_2)$ terms of f collide modulo $x^r - 1$.

6.7. Conclusion

For functions f that can be evaluated "fast" (more specifically, this means that the $(\log T)^3$ term dominates the L term in the complexity bound (12) for geometric progression style algorithms), it may be interesting to switch to the FFT-based approach from this section. As a bonus, one may exploit the Frobenius map whenever we need to work over an extension field of K. Under the assumption that $D = T^{O(1)}$ and that suitable orders $r_1, \ldots, r_l$ as in section 6.2 indeed exist, this leads to the complexity bound

$$O^\flat\left((L + \log T)\, T \left(\frac{\log D}{\log T}\right)^2 \log (qT) \right). \tag{16}$$

We recall that this bound hides an increased constant factor with respect to (12), so geometric progression based algorithms should remain faster whenever $L > (\log T)^2 \log D$. Under the assumption that $D = T^{O(1)}$, we also notice that (16) is always better than (8). However, cyclic extension methods should remain faster in the supersparse case. One noticeable exception concerns sparse interpolation over the rationals, in which case we may use the approach from section 6.5.

7. MULTIVARIATE SPARSE INTERPOLATION

In this section we focus on the case when f is a multivariate polynomial in $K[x_1, \ldots, x_n]$ given as a black box function. In sections 7.1 and 7.2, we first survey the well-known technique of Kronecker segmentation and known multivariate variants of Algorithm 4. In section 7.3, we recall a technique that we introduced in [26] and in section 7.4 we list a few generally useful ideas. In sections 7.5, 7.6, and 7.7 we propose a few new ideas for particular situations.

7.1. Reduction to the univariate case using Kronecker segmentation

One first method for interpolating f is to reduce to the univariate case using Kronecker segmentation. This reduction assumes that we have strict bounds $\deg_{x_1} f < D_1, \ldots, \deg_{x_n} f < D_n$ for the partial degrees of f. Then we may consider the univariate polynomial

$$g(z) = f(z, z^{d_1}, \ldots, z^{d_1 \cdots d_{n-1}})$$

with $\deg g < D_1 \cdots D_n$. We now compute the univariate sparse interpolation of g and retrieve the sparse interpolation of f by mapping each term $g_i z^i$ of g to the term $g_i x_1^{i \operatorname{rem} d_1} x_2^{(i \operatorname{quo} d_1) \operatorname{rem} d_2} \cdots x_n^{i \operatorname{rem} d_1 \cdots d_{n-1}}$ of f.

7.2. Generalizing algorithms based on geometric progressions

Another well known method for multivariate sparse interpolation is to adapt Algorithm 4. Using the notation $x^k = x_1^{k_1} \cdots x_n^{k_n}$ for n-tuples $k \in \mathbb{N}^n$, we may still write

$$f = c_1 x^{e_1} + \cdots + c_t x^{e_t}.$$

For a suitable vector $\omega \in K^r$, we again evaluate f at $\omega^k = \omega_1^k \cdots \omega_n^k$ for $k = 0, \ldots, 2T-1$, and determine the roots $\omega^{-e_1}, \ldots, \omega^{-e_t}$ of Λ as in steps 3 and 4 of Algorithm 4 for the univariate case. The only subtlety is that we need to chose ω in a clever way, so that we can recover the exponents $e_1, \ldots, e_t$ from $\omega^{e_1}, \ldots, \omega^{e_t}$.

In the case when K has characteristic zero, Ben-Or and Tiwari [8] proposed to pick $\omega = (p_1, \ldots, p_n)$, where $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, ... is the sequence of prime numbers; see also [19]. This clearly allows us to quickly recover each e_i from ω^{e_i} through smooth prime factorization. The approach still works for finite fields K of characteristic char $K > p_n^D$. With respect to Kronecker segmentation, the approach is interesting if the total degree D is much smaller than the sum $D_1 + \cdots + D_n$ of the partial degrees. Note that we really need the weighted total degree

$$D' := \max \{(e_i)_1 \log p_1 + \cdots + (e_i)_n \log p_n : 1 \leqslant i \leqslant t\}$$

to be bounded by $D' < \log \operatorname{char} K$.

If K is a finite field, then we may still use a variant of this idea, which partially goes back to [28]. Modulo going to a suitable field extension, we first ensure that K is of the form $K = \mathbb{F}_{q^s}$, where $q \geqslant n$ and $s > D$. For some primitive element $u \in \mathbb{F}_{q^s}$ over $\mathbb{F}_q$ and n pairwise distinct points $a_1, \ldots, a_n \in \mathbb{F}_q$ we then take $\omega = (u - a_1, \ldots, u - a_n)$.

For sparse interpolations on GPUs, modular arithmetic is often most efficient for prime numbers of at most 23 or 31 bits. This may be insufficient for the interpolation of very large polynomials with $t \gg 2^{32}$ and thereby force us to use $K = \mathbb{F}_{q^2}$ instead. In that case, we may take $\omega = (p_1, \ldots, p_k, u - a_1, \ldots, u - a_{n-k})$ and $a_1, \ldots, a_{n-k} \in \mathbb{F}_q$. We next require the total degree of f in $x_1, \ldots, x_k$ to be bounded by D with $p_k^D < q$ and we require f to be linear in $x_{k+1}, \ldots, x_n$.

If one of the above techniques applies, then the complexity analysis is similar to the one for the univariate case. Indeed, after the evaluation step 2 of Algorithm 4, the values $f(1), f(\omega), \ldots, f(\omega^{2T-1})$ are in K, so steps 3, 4, and 6 can be applied without changes, whereas the discrete logarithm computations in step 5 are replaced by cheaper factorizations. Under the assumption that we can find a suitable divisor $r = r_1 r_2$ of $q - 1$ with $r_1 \approx T$ and r_2 smooth, this leads to a complexity bound of the form

$$O^\flat((L + (\log T)^3)\, T \log (qT)). \tag{17}$$

7.3. Multivariate interpolation by packets of coordinates

For $m \in \{1, \ldots, n-1\}$ and a random point $\alpha \in K^m$, consider the specialization

$$\varphi(x_{m+1}, \ldots, x_n) := f(\alpha_1, \ldots, \alpha_m, x_{m+1}, \ldots, x_n).$$

With high probability on $\alpha_1, \ldots, \alpha_m$, the support of φ coincides with the support of f in $x_1, \ldots, x_m$. The knowledge of the support of φ may sometimes help with the computation of the support of f, which in turn facilitates its sparse interpolation. Since φ has less variables, we may recursively use a similar strategy for determining the support of φ. This idea essentially goes back to Zippel [45] in the case when $m = n - 1$ and was applied successfully in [26] in combination with more modern algorithms for sparse interpolation.

Let us show how to implement this strategy in the case when we know a bound $D_{>m}$ for the total degree of f with respect to $x_{m+1}, \ldots, x_n$. We first pick $\alpha = (\alpha_{m+1}, \ldots, \alpha_n)$ in a similar way as ω in subsection 7.2: if char $K = 0$ or char $K > p_{n-m}^{D_{>m}}$, then we take $\alpha := (p_1, \ldots, p_{n-m})$; otherwise, we take $\alpha := (u - a_1, \ldots, u - a_{n-m})$ with the notations from there. The auxiliary function

$$g(x_1, \ldots, x_n) = f(x_1, \ldots, x_m, \alpha_{m+1} x_{m+1}, \ldots, \alpha_n x_n)$$

admits the property that the terms $c_i x^{e_i}$ of f are in one to one correspondence with the terms $c_i \alpha_{m+1}^{e_{i,m+1}} \cdots \alpha_n^{e_{i,n}} x^{e_i}$ of g (where $e_{i,j} := (e_i)_j$). Moreover, knowing both c_i and $c_i \alpha_{m+1}^{e_{i,m+1}} \cdots \alpha_n^{e_{i,n}}$ allows us to compute $e_{i,>m} := (e_{i,m+1}, \ldots, e_{i,n})$ through factorization.

For a suitable element $\omega \in K$ of order $r \geqslant T$ and suitable integers $\kappa_1, \ldots, \kappa_n > 0$, the next idea is to compute

$$\begin{aligned} \Phi(z) &:= \varphi(z^{\kappa_{m+1}}, \ldots, z^{\kappa_n}) \operatorname{rem} (z^r - 1) \\ F(z) &:= f(z^{\kappa_1}, \ldots, z^{\kappa_n}) \operatorname{rem} (z^r - 1) \\ G(z) &:= g(z^{\kappa_1}, \ldots, z^{\kappa_n}) \operatorname{rem} (z^r - 1). \end{aligned}$$

Mimicking Algorithm 4, we take $r = r_1 r_2 \geqslant 2T$ as large as possible under the conditions that $r_1 = O(T)$ and r_2 be smooth (ideally speaking, $r_1 \asymp r_2 \asymp T$). We compute the polynomial Λ, its roots, and the exponents $\kappa \cdot e_1, \ldots, \kappa \cdot e_t$ modulo r as in Algorithm 4, which allows us to recover $F(z)$. We compute $G(z)$ in a similar way (while exploiting the fact that the polynomial Λ is the same in both cases with high probability). Now consider some $i \in \{1, \ldots, t\}$ such that $\kappa \cdot e_i \neq \kappa \cdot e_j$ modulo r and $\kappa_{\leqslant m} \cdot e_{i,\leqslant m} \neq \kappa \cdot e_{j,\leqslant m}$ modulo r for all $j \neq i$. Then we may recover $e_{i,>m}$ from the terms $f_i z^{\kappa \cdot e_i \operatorname{rem} r}$ and $f_i \alpha_{>m}^{e_{i,m}} z^{\kappa \cdot e_i \operatorname{rem} r}$ of $F(z)$ and $G(z)$. Moreover, $\kappa \cdot e_{i,\leqslant m} \operatorname{rem} r = (\kappa \cdot e_i - \kappa \cdot e_{i,>m}) \operatorname{rem} r$ is an exponent of Φ (with high probability). Since φ and Φ are both known, this allows us to recover $e_{i,\leqslant m}$.

Instead of a geometric progression method, we may also opt for the approach from section 6. In that case, we rather take an order $r \geqslant T$ close to T, and we compute $F(z)$ and $G(z)$ using Algorithm 5. With high probability, this only leads to a constant portion of the exponents of f. By varying the choices of r and κ, we may increase this portion until we find all exponents.

This way of interpolating multivariate polynomials by packets of variables is particularly interesting if φ admits significantly less terms than T (so that the interpolation of φ only requires a small portion of the total time) and $D_{>m}$ is small. In that case, we can typically avoid expensive field extensions as considered in sections 5.3, 6.3, and 6.4. If $K = \mathbb{F}_q$ is a finite field with $q \geqslant T$, this leads to a complexity of $O^{\flat}((L + (\log T)^3)\, T \log q)$ for the geometric progression method and $O^{\flat}((L + \log T)\, T \log q)$ when using FFTs (but with a larger constant factor for the dependency on L).

Remark 5. The algorithms in this subsection rely on the prime factoring approach from section 7.2. It is also possible to use Kronecker segmentation for the next packet of variables: see [26]. Notice that Kronecker segmentation is mainly useful in combination with geometric progression style algorithms. It combines less well with the FFT based approach, except when $m = n - 1$ and the coefficients of f with respect to $x_1, \ldots, x_m$ are dense polynomials in x_n.

7.4. Gathering information

In the previous subsections, besides a bound on the total degree D, we have also used bounds on the partial degrees $D_1, \ldots, D_n$ and the total degree $D_{>m}$ with respect to a packet of variables $\{x_{m+1}, \ldots, x_n\}$. More generally, for the sparse interpolation of a large multivariate polynomial f, it is a good idea to first gather some general information about f. This can often be done quickly and the information can be used to select a fast interpolation strategy for the specific polynomial at hand. For instance, in the previous subsection, it may help finding a packet of variables $\{x_{m+1}, \ldots, x_n\}$ that leads to optimal performance.

Let us now describe a few probabilistic methods to gain information about f.

Zero test We may test whether $f = 0$ through one evaluation at a random point $a \in K^n$.

Constant test We may test whether $f \in K$ through one more evaluation at a random point $\beta \in K^n$ with $\beta \neq \alpha$, by checking whether $f(\alpha) = f(\beta)$.

Linearity test We may test whether f is linear through a third evaluation at the point $\gamma = (1-\lambda)\,\alpha + \lambda\,\beta$ for a random $\lambda \in K \setminus \{0,1\}$, by checking whether $f(\gamma) = (1-\lambda)\,f(\alpha) + \lambda f(\beta)$.

Small total degree More generally, if the total degree d of f is small, then we may determine this degree d using $d+1$ evaluations of f: we first set $g(z) := f(\alpha_1 z, \ldots, \alpha_n z)$ for random $\alpha_1, \ldots, \alpha_n \in K \setminus \{0\}$ so that $d = \deg g$. For $\omega \in K^*$ of order $> d$, we then compute $g(\omega^i)$ for $i = 0,1,2,\ldots$ until $G_{i-1}(\omega^i) = g(\omega^i)$ for the unique polynomial G_{i-1} of degree $i-1$ with $g(\omega^j) = G_{i-1}(\omega^j)$ for $j = 0, \ldots, i-1$. At that point, we have $i = d$.

Small partial degrees The above method generalizes to the case when we wish to determine the degree of f with respect to all variables in a subset S of $\{x_1,...,x_n\}$. Indeed, modulo renaming variables, we may assume $S = \{x_1, \ldots, x_m\}$. We then take $g(z) := f(\alpha_1 z, \ldots, \alpha_m z, \alpha_{m+1}, \ldots, \alpha_n)$.

Number of terms We can obtain a rough estimate for the number t of terms of f as follows: for orders r that increase geometrically and random integers $\kappa_1, \ldots, \kappa_n > 0$, we compute $F(z) := f(z^{\kappa_1}, \ldots, z^{\kappa_n}) \bmod (z^r - 1)$. As soon as r exceeds $t \log t$, it becomes likely that F contains less than r terms. This may be used to determine a rough estimate of t using $O(t \log t)$ instead of $O(t)$ evaluations.

Active variables We may recursively compute the set of variables x_i that occur in f as follows. If f is constant, then we return $\emptyset$. If f is not constant and $n = 1$, then we return $\{x_1\}$. Otherwise, we pick a random $\alpha \in K$ and consider $g(x_1, \ldots, x_m) := f(x_1, \ldots, x_m, \alpha_{m+1}, \ldots, \alpha_n)$ with $m := \lfloor n/2 \rfloor$, as well as $h(x_{m+1}, \ldots, x_n) := f(\alpha_1, \ldots, \alpha_m, x_{m+1}, \ldots, x_n)$. We recursively apply the method to g and h, and return the union of the output sets. This method uses at most $\lceil \log_2 n \rceil s$ evaluations of f for an output set of size s.

Linear cliques A *linear clique* is a subset $S \subseteq \{x_1, \ldots, x_n\}$ of variables such that f is linear with respect S. Is it possible to quickly determine a maximal linear clique? Setting $X = \{x_i : \deg f_{x_i} \leqslant 1\}$ we must clearly have $S \subseteq X$ and we may always put all inactive variables inside S. Although we do not have an algorithm to compute a maximal linear clique, the following randomized algorithm should be able to produce reasonably large ones for practical purposes (especially if we run it several times). Starting with $S := \emptyset$, we iterate over all elements $x_i \in X$ in a random order, and add a new x_i to S whenever f remains linear with respect to $S \cup \{x_i\}$. Notice that this algorithm only requires $3\,|X|$ evaluations of f.

Low degree packets More generally, we may use a variant of the linear clique algorithm to find large packets of variables $S \subseteq \{x_1, \ldots, x_n\}$ with respect to which f admits a small total degree. This is the kind of packets that we need in section 7.3 (by taking $\{x_{m+1}, \ldots, x_n\} := S$ after a suitable permutation of indices).

More precisely, for a finite field $K = \mathbb{F}_q$ of size q, assume that we wish to compute a packet S of size s (as large as possible) such that the total degree d_S of f with respect to S satisfies $p_s^{d_S} < q$. During the algorithm, we maintain a table that associates a degree δ_i to each variable x_i. The table is initialized with $\delta_i := \deg_{x_i} f$ for each i. Starting with $S := \emptyset$ we now repeat the following steps. We reorder indices such that $S = \{x_1, \ldots, x_s\}$ and $\delta_{s+1} \leqslant \cdots \leqslant \delta_n$. For $i = s+1, s+2, \ldots$, we replace $\delta_i := \deg_{S \cup \{x_i\}} f$ and stop as soon as $i = n$ or $\delta_j \leqslant \delta_{i+1}$ for some $j \in \{s+1, \ldots, i\}$. We then determine the index $j \in \{s+1, \ldots, i\}$ for which δ_j is minimal. If $p_{s+1}^{\delta_j} < q$, then we add x_j to S and continue; otherwise, we abort the main loop and return S.

During the loop, we may also remove all variables x_i for which δ_i exceeds $\log q / \log p_{s+1}$. With this optimization, the number of evaluations of f always remains bounded by $O(n^2 \log q)$.

7.5. Packets of total degree one

Let us now consider the interesting special case when there exist a non-trivial linear clique $\{x_{m+1}, \ldots, x_n\}$ and assume that we know the support of f with respect to $x_1, \ldots, x_m$. Then the derivation strategy from section 3.2 yields an efficient interpolation method for f, even if char K is small. Indeed, in that case, we may write

$$f = g_0(x_1, \ldots, x_m) + g_1(x_1, \ldots, x_m)\, x_{m+1} + \cdots + g_{n-m}(x_1, \ldots, x_m)\, x_n,$$

where $g_{m+i} = \partial f / \partial x_{m+i}$ for $i = 1, \ldots, n-m$. The idea is to jointly interpolate $g_0, g_1, \ldots, g_{n-m}$ using the fact that the vector $v := (g_0, g_1, \ldots, g_{n-m})$ can be evaluated fast using automatic differentiation. As usual, we may either rely on a geometric progression style approach or on more direct FFTs.

When using a geometric progression style approach, we assume that $T^2 \prec q$ and let $\omega \in K$ denote a primitive $(q-1)$-th root of unity. We pick random integers $\kappa_1, \ldots, \kappa_m > 0$ and set $V(z) := v(z^{\kappa_1}, \ldots, z^{\kappa_m})$. With high probability, the known exponents $e_1, \ldots, e_u$ of V are pairwise distinct modulo $q-1$. We now compute $V(1), \ldots, V(\omega^{u-1})$ and interpolate the coefficients of V using transposed multipoint interpolation. This yields V and v in time

$$O^\flat((L + (\log u)^2)\, u \log q). \tag{18}$$

When using FFTs, we rather take an r-th primitive root of unity with $r \mid (q-1)$ close to u and only obtain a constant proportion of the coefficients of v; applying the same method for different values of r, this leads to a runtime of $O^\flat((L + \log u)\, u \log q)$, with a higher constant factor with respect to L.

Example 6. Consider the sparse interpolation of the determinant $f = \det$ of an $n \times n$ matrix with generic coefficients $(x_{i,j})_{1 \leqslant i,j \leqslant n}$. Then the variables $x_{1,1}, \ldots, x_{1,n}$ form a (maximal) linear clique. When specializing these variables by random elements in K, we obtain a new function φ for which the variables $x_{2,1}, \ldots, x_{2,n}$ form a (maximal) linear clique. And so on for the successive rows of the matrix. Notice that the specialization of all variables $x_{i,j}$ with $i \leqslant k$ has a support of size $n!/k!$.

Using the geometric progression strategy of complexity (18) in a recursive manner, it follows that we can compute the sparse interpolation of f in time

$$O^\flat\left(\sum_{1 \leqslant k \leqslant n} \left(\left(n^3 + \left(\log \frac{n!}{k!}\right)^2\right) \frac{n!}{k!}\right) \log q\right) = O^\flat((n^3 + (\log n!)^2)\, n! \log q) = O^\flat(n^3 n! \log q).$$

Notice that the traditional geometric progression method (of complexity (17)) takes time

$$O^\flat((n^3 + (\log n!)^3)\, n! \log q) = O^\flat(n^3 (\log n)^3 n! \log q).$$

7.6. Multivariate FFTs

Instead of evaluating $F(z) := f(z^{\kappa_1}, \ldots, z^{\kappa_n}) \bmod (z^r - 1)$ using Algorithm 5 for random integers $\kappa_1, \ldots, \kappa_n > 0$ and a suitable modulus $r \mid (q-1)$, the multivariate setting also allows us to consider multivariate FFTs. To do so, we use moduli $r_1, \ldots, r_m \mid (q-1)$ and an integer matrix $(\kappa_{i,j})_{i \leqslant n, j \leqslant m}$, and then take

$$F(z_1, \ldots, z_m) := f(z_1^{\kappa_{1,1}} \cdots z_m^{\kappa_{1,m}}, \ldots, z_1^{\kappa_{n,1}} \cdots z_m^{\kappa_{n,m}}) \bmod (z_1^{r_1} - 1, \ldots, z_m^{r_m} - 1).$$

Moreover, the r_i and $\kappa_{i,j}$ should be chosen in such a way that $r := r_1 \cdots r_m$ is close to T, the linear map

$$\begin{aligned} \mathbb{Z}^n &\xrightarrow{\psi} \mathbb{Z}/r_1\mathbb{Z} \times \cdots \times \mathbb{Z}/r_m\mathbb{Z} \\ (e_1, \ldots, e_n) &\longmapsto (e_1\kappa_{1,1} + \cdots + e_n\kappa_{n,1}, \ldots, e_1\kappa_{1,m} + \cdots + e_n\kappa_{n,m}) \end{aligned}$$

is surjective, and the exponents in the support of f are mapped to essentially random elements in $\operatorname{im}\psi$. This generalization admits the advantage that the r_i do not need to be coprime, which may help to avoid working over large extensions $\mathbb{F}_{q^s}$ if $D \gg T$. It would be interesting to generalize the Frobenius FFT to this multivariate setting.

7.7. Symmetries

Is it possible to speed up the interpolation if f admits symmetries? As a simple example, let us consider the trivariate case with $f(x_1, x_2, x_3) = f(x_2, x_1, x_3)$. It is easy to detect such a symmetry by checking whether $f(\alpha_1, \alpha_2, \alpha_3) = f(\alpha_2, \alpha_1, \alpha_3)$ for a random point $\alpha \in K^3$. When using multivariate FFTs as in the previous subsection, the idea is to pick parameters r_i and $\kappa_{i,j}$ of the form

$$F(z_1, z_2, z_3) := f(z_1 z_3^{\kappa_{1,3}}, z_2 z_3^{\kappa_{1,3}}, z_1^{\kappa_{3,1}} z_2^{\kappa_{3,1}} z_3^{\kappa_{3,3}}) \text{ rem } (z_1^{r_1} - 1, z_2^{r_1} - 1, z_3^{r_3} - 1),$$

so that F is still symmetric with respect to z_1 and z_2. Now let $\omega_1, \omega_3 \in K$ be primitive roots of unity of orders r_1 and r_3. In order to evaluate of F at all points $(\omega_1^{i_1}, \omega_1^{i_2}, \omega_3^{i_3})$ with $i_1, i_2 = 0, \ldots, r_1 - 1$ and $i_3 = 0, \ldots, r_3 - 1$, it suffices to consider the points with $i_1 \leqslant i_2$, thereby saving a factor of almost two. We may recover the coefficients of F using an inverse DFT. This can again be done almost twice as efficiently as for non-symmetric polynomials, using a method from [24]. Overall, we save a factor of almost two when computing sparse evaluations using the FFT method. This approach should generalize to more general types of symmetries, as the ones considered in [24].

7.8. Conclusion

In order to interpolate a multivariate polynomial f, it is recommended to first gather useful information about f, as described in section 7.4. With this information at hand, we may then opt for a most appropriate interpolation strategy:

- If f admits special properties, then we may wish to apply a dedicated algorithm, such as the linear clique strategy from section 7.5 or the symmetric interpolation strategy from from section 7.7.
- Whenever $D_1 \cdots D_n < q - 1$, Kronecker segmentation can be combined with any of the univariate interpolation methods in order to obtain an efficient algorithm for the interpolation of f.
- If $D_1 \cdots D_n \geqslant q - 1$, but f admits a reasonably small total degree, then we may use a variant of the geometric progression style algorithm from section 7.2 in order to interpolate f.

- If f is normally sparse, but too large for one of the above strategies to apply, then interpolation by packets is a powerful tool. It typically allows us to reduce to the most favorable cases from sections 5 and 6 when we do not need any field extensions of $\mathbb{F}_q$ (except when $q<T$ or when $q-1$ admits not enough small prime divisors). As in the univariate case, we may then opt for a geometric progression style approach if $(\log T)^3 = O(L)$ and for an FFT based approach if $L = o((\log T)^3)$.
- If f is supersparse, then we may use the strategy from section 6.5 over the rationals and fall back on a cyclic extension style algorithm when K is a given finite field.

BIBLIOGRAPHY

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[2] A. Arnold, M. Giesbrecht, and D. S. Roche. Sparse interpolation over finite fields via low-order roots of unity. In *ISSAC '14: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 27–34. ACM Press, 2014.

[3] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse multivariate polynomial interpolation of straight-line programs. *J. Symbolic Comput.*, 75:4–24, 2016.

[4] A. Arnold and D. S. Roche. Multivariate sparse interpolation using randomized Kronecker substitutions. In *ISSAC '14: Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 35–42. ACM Press, 2014.

[5] R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bltn Mathcal Biology*, 51(1):125–131, 1989.

[6] R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes, II. *Proc. of the London Math. Soc.*, 83(3):532–562, 2001.

[7] W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[8] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309. ACM Press, 1988.

[9] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC '03: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 37–44. ACM Press, 2003.

[10] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1(3):259–295, 1980.

[11] J. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 121–128. ACM Press, 1989.

[12] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comp.*, 36(154):587–592, 1981.

[13] A. Díaz and E. Kaltofen. FOXFOX: a system for manipulating symbolic objects in black box representation. In *ISSAC '98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, pages 30–37. ACM Press, 1998.

[14] T. S. Freeman, G. M. Imirzian, E. Kaltofen, and Y. Lakshman. DAGWOOD: a system for manipulating polynomials given by straight-line programs. *ACM Trans. Math. Software*, 14:218–240, 1988.

[15] S. Garg and É. Schost. Interpolation of polynomials given by straight-line programs. *Theor. Comput. Sci.*, 410(27-29):2659–2662, 2009.

[16] M. Giesbrecht and D. S. Roche. Diversification improves interpolation. In *ISSAC '11: Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, pages 123–130. ACM Press, 2011.

[17] B. Grenet, J. van der Hoeven, and G. Lecerf. Randomized root finding over finite fields using tangent Graeffe transforms. In *ISSAC '15: Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 197–204. ACM Press, 2015.

[18] B. Grenet, J. van der Hoeven, and G. Lecerf. Deterministic root finding over finite fields using Graeffe transforms. *Appl. Algebra Engrg. Comm. Comput.*, 27(3):237–257, 2016.

[19] D. Y. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pages 166–172. IEEE Computer Society, 1987.

[20] D. Harvey and J. van der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. Technical Report, HAL, 2019. http://hal.archives-ouvertes.fr/hal-02070816.

[21] D. Harvey, J. van der Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. *J. ACM*, 63(6), 2017. Article 52.

[22] J. Heintz and C.-P. Schnorr. Testing polynomials which are easy to compute. In *Logic and algorithmic (Zurich, 1980)*, volume 30 of *Monograph. Enseign. Math.*, pages 237–254. Geneva, 1982. Univ. Genève.

[23] J. van der Hoeven and R. Larrieu. The Frobenius FFT. In M. Burr, editor, *ISSAC '17: Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 437–444. ACM Press, 2017.

[24] J. van der Hoeven, R. Lebreton, and É. Schost. Structured FFT and TFT: symmetric and lattice polynomials. In *ISSAC '13: Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 355–362. ACM Press, 2013.

[25] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Comput.*, 50:227–254, 2013.

[26] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation in practice. *ACM Commun. Comput. Algebra*, 48(3/4):187–191, 2015.

[27] J. Hu and M. B. Monagan. A fast parallel sparse polynomial GCD algorithm. In *ISSAC '16: Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 271–278. ACM Press, 2016.

[28] M. A. Huang and A. J. Rao. Interpolation of sparse multivariate polynomials over large finite fields with applications. In *SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 508–517. Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[29] Q. L. Huang and X. S. Gao. Sparse Polynomial Interpolation with Finitely Many Values for the Coefficients. In V. Gerdt, V. Koepf, W. Seiler, and E. Vorozhtsov, editors, *Computer Algebra in Scientific Computing. 19th International Workshop, CASC 2017, Beijing, China, September 18-22, 2017, Proceedings.*, volume 10490 of *Lecture Notes in Computer Science*. Springer, Cham, 2017.

[30] Q.-L. Huang. Sparse polynomial interpolation over fields with large or zero characteristic. In *Proc. ISSAC '19*, pages 219–226. ACM, 2019.

[31] M. Javadi and M. Monagan. Parallel sparse polynomial interpolation over finite fields. In M. Moreno Maza and J.-L. Roch, editors, *PASCO '10: Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 160–168. ACM Press, 2010.

[32] E. Kaltofen. Computing with polynomials given by straight-line programs I: greatest common divisors. In *STOC '85: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 131–142. ACM Press, 1985.

[33] E. L. Kaltofen. Fifteen years after DSC and WLSS2 what parallel computations I do today: invited lecture at PASCO 2010. In *PASCO '10: Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 10–17. ACM Press, 2010.

[34] E. Kaltofen, Y. N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *ISSAC '90: Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 135–139. New York, NY, USA, 1990. ACM Press.

[35] E. Kaltofen and W. Lee. Early termination in sparse interpolation algorithms. *J. Symbolic Comput.*, 36(3):365–400, 2003.

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

[37] E. Kaltofen and L. Yagati. Improved sparse multivariate polynomial interpolation algorithms. In P. Gianni, editor, *Symbolic and algebraic computation. International symposium ISSAC '88. Rome, Italy, July 4–8, 1988*, volume 358, pages 467–474. Springer-Verlag, 1989.

[38] R. Moenck. Fast computation of GCDs. In *STOC '73: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pages 142–171. ACM Press, 1973.

[39] H. Murao and T. Fujise. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *JSC*, 21:377–396, 1996.

[40] M. Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Ann. Inst. Stat. Math.*, 10(1):29–35, 1959.

[41] R. Prony. Essai expérimental et analytique sur les lois de la dilatabilité des fluides élastiques et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures. *J. de l'École Polytechnique Floréal et Plairial, an III*, 1(cahier 22):24–76, 1795.

[42] J.-J. Risler and F. Ronga. Testing polynomials. *J. Symbolic Comput.*, 10(1):1–5, 1990.

[43] D. S. Roche. What can (and can't) we do with sparse polynomials? In C. Arreche, editor, *ISSAC '18: Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, pages 25–30. ACM Press, 2018.

[44] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica*, 7:395–398, 1977.

[45] R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *Symbolic and Algebraic Computation. Eurosam '79, An International Symposium on Symbolic and Algebraic Manipulation, Marseille, France, June 1979*, volume 72 of *Lect. Notes Comput. Sci.*, pages 216–226. Springer-Verlag, 1979.