Fast Hermite interpolation and evaluation over finite fields of characteristic two

Nicholas Coxon

▶ **To cite this version:**

Nicholas Coxon. Fast Hermite interpolation and evaluation over finite fields of characteristic two. 2018. hal-01827583v1

HAL Id: hal-01827583

https://hal.science/hal-01827583v1

Preprint submitted on 2 Jul 2018 (v1), last revised 5 Sep 2019 (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.

FAST HERMITE INTERPOLATION AND EVALUATION OVER FINITE FIELDS OF CHARACTERISTIC TWO

NICHOLAS COXON

Abstract. This paper presents new fast algorithms for Hermite interpolation and evaluation over finite fields of characteristic two. The algorithms reduce the Hermite problems to instances of the standard multipoint interpolation and evaluation problems, which are then solved by existing fast algorithms. The reductions are simple to implement and free of multiplications, allowing low overall multiplicative complexities to be obtained. The algorithms are suitable for use in encoding and decoding algorithms for multiplicity codes.

1. Introduction

Hermite interpolation is the problem of computing the coefficients of a polynomial given the values of its derivatives up to a given order at one or more evaluation points. The inverse problem, that of evaluating the derivatives of the polynomial when given its coefficients, is sometimes referred to as Hermite evaluation. Over fields of positive characteristic p, the ith formal derivative vanishes identically for $i \geq p$. Consequently, it usual to consider Hermite interpolation and evaluation with respect to the Hasse derivative over fields of small positive characteristic.

For now, let $\mathbb{F}$ simply denote a field. Then, for $i \in \mathbb{N}$, the map $D^i : \mathbb{F}[x] \to \mathbb{F}[x]$ that sends $F \in \mathbb{F}[x]$ to the coefficient of y^i in $F(x+y) \in \mathbb{F}[x][y]$ is called the ith Hasse derivative on $\mathbb{F}[x]$. For distinct evaluation points $\omega_0, \dots, \omega_{n-1} \in \mathbb{F}$ and positive integer multiplicities $\ell_0, \dots, \ell_{n-1}$, the Hermite interpolation problem over $\mathbb{F}$ asks that we compute the coefficients of a polynomial $F \in \mathbb{F}[x]$ of degree less than $\ell = \ell_0 + \cdots + \ell_{n-1}$ when given $(D^i F)(\omega_j)$ for $j \in \{0, \dots, \ell_i - 1\}$ and $i \in \{0, \dots, n-1\}$. The corresponding instance of the Hermite evaluation problem asks that we use the coefficients of F to compute the ℓ derivatives of the interpolation problem. Different versions of the problems specify different bases on which the polynomials are required to be represented. In this paper, the problems are considered with respect to the monomial basis $\{1, x, x^2, \dots\}$ of $\mathbb{F}[x]$ only.

The boundary case $\ell_0 = \cdots = \ell_{n-1} = 1$ corresponds to standard multipoint interpolation and evaluation, allowing both problems to be solved with $\mathcal{O}(\mathsf{M}(\ell) \log \ell)$ operations in $\mathbb{F}$ by the use of remainder trees and fast Chinese remainder algorithms [17, 30, 7, 9, 8, 3, 37] (see also [41, Chapter 10]). Here, $\mathsf{M}(\ell)$ denotes the number of operations required to multiply two polynomials in $\mathbb{F}[x]$ of degree less

INRIA and Laboratoire d'Informatique de l'École polytechnique, Palaiseau, France.
E-mail address: nicholas.coxon@inria.fr.

Date: July 2, 2018.
Key words and phrases. Hermite interpolation, Hermite evaluation, multiplicity codes.
This work was supported by Nokia in the framework of the common laboratory between Nokia Bell Labs and INRIA.

than ℓ, which may be taken to be in $\mathcal{O}(\ell(\log \ell) \log\log \ell)$ [34, 33, 12]. The complexity of solving the standard interpolation and evaluation problems reduces to $\mathcal{O}(\mathsf{M}(\ell))$ operations when the evaluation points form a geometric progression [10]. Similarly, fast Fourier transform (FFT) [15] based interpolation and evaluation offer complexities as low as $\mathcal{O}(\ell \log \ell)$ operations on certain special sets of evaluation points.

For the opposing boundary case of $n = 1$, the Hermite interpolation and evaluation problems both reduce to computing Taylor expansions. Indeed, it follows directly from the definition of Hasse derivatives that

$$F = \sum_{i \in \mathbb{N}} (D^i F)(\omega)(x - \omega)^i \quad \text{for } F \in \mathbb{F}[x] \text{ and } \omega \in \mathbb{F}. \tag{1.1}$$

Consequently, Hermite interpolation and evaluation at a single evaluation point can be performed in $\mathcal{O}(\mathsf{M}(\ell) \log \ell)$ operations in general [7, 39, 40], $\mathcal{O}(\mathsf{M}(\ell))$ operations if $(\ell - 1)!$ is invertible in the field [1, 38] (see also [40, 6]), and $\mathcal{O}(\ell \log \ell)$ operations if the field has characteristic equal to two [19].

The first quasi-linear time algorithms for solving the general Hermite problems were proposed by Chin [14]. Truncating the Taylor expansion (1.1) after i terms gives the residue of degree less than i of F modulo $(x - \omega)^i$. Based on this observation, Chin's evaluation algorithm begins by using a remainder tree to compute the residues of the input polynomial modulo $(x - \omega_i)^{\ell_i}$ for $i \in \{0, \ldots, n-1\}$. The Taylor expansion of each residue at its corresponding evaluation point is then computed to obtain the truncated Taylor expansion of the input polynomial. The interpolation problem can be solved by reversing these steps, with the residues combined by a fast Chinese remainder algorithm. It follows that the general Hermite interpolation and evaluation problems may be solved with $\mathcal{O}(\mathsf{M}(\ell) \log \ell)$ operations [14, 31] (see also [6, 32]).

In this paper, we present new algorithms for Hermite interpolation and evaluation over finite fields of characteristic two. The algorithms require the set of evaluation points to equal the field itself, and their corresponding multiplicities to be balanced, with $|\ell_i - \ell_j| \leq 1$ for $i \neq j$. While not solving the general interpolation and evaluation problems over these fields, the algorithms are suitable for use in multivariate Hermite interpolation and evaluation algorithms [16], encoding and decoding algorithms for multiplicity codes [22, 16] and the codes of Wu [43], and private information retrieval protocols based on these codes [42, 2].

Over a characteristic two finite field of order q, the restricted problems may be solved with $\mathcal{O}(\mathsf{M}(\ell) \log q + \ell \log \ell/q)$ operations by existing algorithms. The algorithms presented in this paper yield the same complexities, but benefit by their simplicity and the low number of multiplications they perform. When ℓ is a multiple of q, as occurs in some encoding and decoding contexts, the Hermite interpolation algorithm presented here performs ℓ/q standard interpolations over the q evaluation points, followed by $\mathcal{O}(\ell \log \ell/q)$ additions. The Hermite evaluation algorithm performs $\mathcal{O}(\ell \log \ell/q)$ additions, followed by ℓ/q standard evaluations over the q points. Using the generic bound of $\mathcal{O}(\mathsf{M}(q) \log q)$ operations for solving the standard problems leads to the above bound on solving the Hermite problems in this special case.

We are not prohibited from using faster FFT-based interpolation and evaluation algorithms to solve the standard problems when they are supported by the field. Moreover, we have the option of using "additive FFT" algorithms [11, 29, 19, 5,

4, 26, 25, 24, 13], which are specific to characteristic two finite fields and allow evaluation and interpolation over the q points of the field to be performed with $\mathcal{O}(q \log^2 q)$ or $\mathcal{O}(q(\log q) \log\log q)$ additions, depending on the degree of the field, and $\mathcal{O}(q \log q)$ multiplications. With these algorithms and ℓ a multiple of q, the Hermite interpolation and evaluation algorithms perform $\mathcal{O}(\ell(\log^2 q + \log \ell/q))$ or $\mathcal{O}(\ell((\log q) \log\log q + \log \ell/q))$ additions, and only $\mathcal{O}(\ell \log q)$ multiplications.

When ℓ is not a multiple of q, the Hermite interpolation and evaluation algorithms still perform $\lceil \ell/q \rceil - 1$ standard interpolations or evaluations over the q evaluation points, but each must also solve one instance of a slightly generalised version of the corresponding standard problem. However, these more general problems reduce to the standard problems at the cost of $\mathcal{O}(\mathsf{M}(q))$ operations for performing one division with remainder of polynomials of degree less than q (see [41, Section 9.1]). Consequently, the algorithms retain their simplicity in this case.

The reduction from Hermite to standard problems is provided in Section 3, where we develop divide-and-conquer algorithms for solving the Hermite interpolation and evaluation problems when ℓ/q is a power of two. The problems for arbitrary ℓ can be reduced to this special case by zero padding. However, this approach almost doubles the size of the initial problem when ℓ/q is slightly larger than a power of two, leading to large jumps in complexity. Instead, in Sections 4 and 5 we address the problems for arbitrary ℓ by transferring across ideas from pruned and truncated FFT algorithms [28, 35, 36, 20, 21, 23], which are used to smooth similar unwanted jumps in the complexities of FFT-based evaluation and interpolation schemes. We are consequently able to solve the Hermite interpolation and evaluation problems with better complexity than obtained by zero padding.

2. Properties of Hasse derivatives

We begin by recalling some basic properties of Hasse derivatives.

Lemma 2.1. *Let $F, G \in \mathbb{F}[x]$, $\alpha, \beta, \omega \in \mathbb{F}$ and $i \in \mathbb{N}$. Then*

1. $D^i(\alpha F + \beta G) = \alpha(D^i F) + \beta(D^i G)$,
2. $(D^i F)(\omega)$ *is equal to the coefficient of* x^i *in* $F(x + \omega)$,
3. $(D^j F)(\omega) = 0$ *for* $j \in \{0, \dots, i-1\}$ *if and only if* $(x - \omega)^i$ *divides* F,
4. $D^i x^k = \binom{k}{i} x^{k-i}$ *for* $k \in \mathbb{N}$, *and*
5. $D^i \circ D^j = \binom{i+j}{i} D^{i+j}$ *for* $j \in \mathbb{N}$.

Properties (1) and (2) of Lemma 2.1 follow readily from the definition of Hasse derivatives provided in the introduction. Property (3) follows from Property (2). Property (4) follows from the definition of Hasse derivatives and the binomial theorem. Property (5) follows from Properties (1) and (4), and the binomial identity

$$\binom{k-j}{i}\binom{k}{j} = \binom{i+j}{i}\binom{k}{i+j} \quad \text{for } i, j, k \in \mathbb{N}.$$

For $\ell > 0$, let $\mathbb{F}[x]_\ell$ denote the space of polynomials in $\mathbb{F}[x]$ that have degree less than ℓ. Then existence and uniqueness for the general Hermite interpolation problem is provided by the following lemma.

Lemma 2.2. *Let $\omega_0, \dots, \omega_{n-1} \in \mathbb{F}$ be distinct, $\ell_0, \dots, \ell_{n-1}$ be positive integers, and $\ell = \ell_0 + \cdots + \ell_{n-1}$. Then given elements $h_{i,j} \in \mathbb{F}$ for $i \in \{0, \dots, \ell_j - 1\}$ and $j \in \{0, \dots, n-1\}$, there exists a unique polynomial $F \in \mathbb{F}[x]_\ell$ such that $(D^i F)(\omega_j) = h_{i,j}$ for $i \in \{0, \dots, \ell_j - 1\}$ and $j \in \{0, \dots, n-1\}$.*

Lemma 2.2 follows from Property (3) of Lemma 2.1, which implies that the kernel of the linear map from $\mathbb{F}[x]_\ell$ to $\mathbb{F}^\ell$ given by $F \mapsto ((D^i F)(\omega_j))_{0 \leq i < \ell_j, 0 \leq j < n}$ can only contain multiples of the degree ℓ polynomial $\prod_{j=0}^{n-1}(x - \omega_j)^{\ell_j}$, and must therefore be trivial.

3. Strategy over finite fields of characteristic two

Hereafter, we assume that $\mathbb{F}$ is finite of characteristic two. Let q denote the order of the field, and enumerate its elements as $\omega_0, \ldots, \omega_{q-1}$. Define $i \operatorname{div} j = \lfloor i/j \rfloor$ and $i \bmod j = i - \lfloor i/j \rfloor j$ for $i, j \in \mathbb{Z}$ such that j is nonzero. Then the Hermite interpolation problem we consider in the remainder of the paper can be stated as follows: given $(h_0, \ldots, h_{\ell-1}) \in \mathbb{F}^\ell$, compute the vector $(f_0, \ldots, f_{\ell-1}) \in \mathbb{F}^\ell$ such that $F = \sum_{i=0}^{\ell-1} f_i x^i$ satisfies $(D^{i \operatorname{div} q} F)(\omega_{i \bmod q}) = h_i$ for $i \in \{0, \ldots, \ell - 1\}$. The Hermite evaluation problem we consider is the inverse problem, asking that we compute the vector $((D^{i \operatorname{div} q} F)(\omega_{i \bmod q}))_{0 \leq i < \ell}$ when given the coefficient vector of $F \in \mathbb{F}[x]_\ell$. We call ℓ the length of an instance of either problem, and observe that if $\ell \leq q$, then the problems reduce to standard multipoint interpolation and evaluation with evaluation points $\omega_0, \ldots, \omega_{\ell-1}$.

In this section, we introduce the main elements of our algorithms by temporarily limiting our attention to instances of length $2^n q$ for some $n \in \mathbb{N}$. The algorithms take on their simplest form in this case, with each applying a simple reduction from the length $2^n q$ problem to two problems of length $2^{n-1} q$. Proceeding recursively, both algorithms ultimately reduce to problems of length q, which are then solved by existing standard interpolation and evaluation algorithms. The reductions employed by the algorithms are provided by the following lemma.

Lemma 3.1. *Let $n \in \mathbb{N}$ be nonzero, $F_0, F_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and*

$$F = F_1(x^q - x)^{2^{n-1}} + F_0. \tag{3.1}$$

Then for $\omega \in \mathbb{F}$ and $i \in \{0, \ldots, 2^n - 1\}$,

$$\big(D^i F\big)(\omega) = \begin{cases} \big(D^i F_0\big)(\omega) & \text{if } i < 2^{n-1}, \\ \big(D^{i-2^{n-1}}(F_1 + D^{2^{n-1}} F_0)\big)(\omega) & \text{otherwise.} \end{cases}$$

Proof. Let $n \in \mathbb{N}$ be nonzero, $F_0, F_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and define F by (3.1). Then

$$F(x + \omega) = F_1(x + \omega)x^{2^{n-1}q} + F_1(x + \omega)x^{2^{n-1}} + F_0(x + \omega)$$

for $\omega \in \mathbb{F}$. Consequently, as $2^{n-1}q \geq 2^n$, Property (2) of Lemma 2.1 implies that

$$\big(D^i F\big)(\omega) = \begin{cases} \big(D^i F_0\big)(\omega) & \text{if } i < 2^{n-1}, \\ \big(D^{i-2^{n-1}} F_1\big)(\omega) + \big(D^i F_0\big)(\omega) & \text{otherwise,} \end{cases}$$

for $\omega \in \mathbb{F}$ and $i \in \{0, \ldots, 2^n - 1\}$. Therefore, linearity of Hasse derivatives implies that the lemma will follow if we can show that $D^i = D^{i-2^{n-1}} \circ D^{2^{n-1}}$ for $i \in \{2^{n-1}, \ldots, 2^n - 1\}$. To this end, we use Lucas' lemma [27, p. 230] (see also [18]), which states that

$$\binom{u}{v} \equiv \binom{u \operatorname{div} 2^r}{v \operatorname{div} 2^r}\binom{u \bmod 2^r}{v \bmod 2^r} \pmod 2 \quad \text{for } u, v, r \in \mathbb{N}. \tag{3.2}$$

By combining Lucas' lemma with Property (5) of Lemma 2.1, we find that

$$D^{i-2^{n-1}} \circ D^{2^{n-1}} = \binom{2^{n-1} + (i - 2^{n-1})}{2^{n-1}} D^i = \binom{1}{1}\binom{i - 2^{n-1}}{0} D^i = D^i$$

for $i \in \{2^{n-1}, \ldots, 2^n - 1\}$. □

Given a vector $(h_0, \ldots, h_{2^n q-1}) \in \mathbb{F}^{2^n q}$ that defines an instance of the Hermite interpolation problem, our algorithm recursively computes the corresponding polynomial $F \in \mathbb{F}[x]_{2^n q}$ as follows. If $n = 0$, then we are in the base case of the recursion, and F is recovered by a standard interpolation algorithm. If $n \geq 1$, then the algorithm is recursively called on $(h_0, \ldots, h_{2^{n-1}q-1})$ and $(h_{2^{n-1}q}, \ldots, h_{2^n q-1})$. Lemmas 2.2 and 3.1 imply that the recursive calls return F_0 and $F_1 + D^{2^{n-1}} F_0$, where F_0 and F_1 are the unique polynomials in $\mathbb{F}[x]_{2^{n-1}q}$ that satisfy (3.1). Thus, the algorithm next recovers F_1 by computing $D^{2^{n-1}} F_0$ and adding it to $F_1 + D^{2^{n-1}} F_0$. Finally, F is computed by expanding (3.1) as

$$F = F_1 x^{2^{n-1}q} + F_1 x^{2^{n-1}} + F_0. \tag{3.3}$$

Given a polynomial $F \in \mathbb{F}[x]_{2^n q}$, the evaluation algorithm uses a standard evaluation algorithm in its base case of $n = 0$, and if $n \geq 1$, then it simply reverses the steps of the interpolation algorithm by first computing F_0 and F_1, then $F_1 + D^{2^{n-1}} F_0$, and finally recursively evaluating F_0 and $F_1 + D^{2^{n-1}} F_0$. In both algorithms, the following lemma is used to compute derivatives.

Lemma 3.2. *Let $n \in \mathbb{N}$ be nonzero and $F = \sum_{i=0}^{2^{n-1}q-1} f_i x^i \in \mathbb{F}[x]$. Then*

$$D^{2^{n-1}} F = \sum_{i=0}^{q/2-1} x^{2^n i} \sum_{j=0}^{2^{n-1}-1} f_{2^{n-1}(2i+1)+j} x^j. \tag{3.4}$$

Proof. Let $n \in \mathbb{N}$ be nonzero and $F = \sum_{i=0}^{2^{n-1}q-1} f_i x^i \in \mathbb{F}[x]$. Then Property (4) of Lemma 2.1 and Lucas' lemma, in the form of (3.2), imply that

$$D^{2^{n-1}} x^{2^{n-1}(2i+b)+j} = \binom{2i+b}{1}\binom{j}{0} x^{2^{n-1}(2i+b-1)+j} = b x^{2^{n-1}(2i+b-1)+j}$$

for $b \in \{0, 1\}$, $i \in \mathbb{N}$ and $j \in \{0, \ldots, 2^{n-1} - 1\}$. Therefore, writing F in the form

$$F = \sum_{b=0}^{1} \sum_{i=0}^{q/2-1} \sum_{j=0}^{2^{n-1}-1} f_{2^{n-1}(2i+b)+j} x^{2^{n-1}(2i+b)+j}$$

and applying $D^{2^{n-1}}$ to each of its terms yields (3.4). □

4. Evaluation algorithm

To solve the Hermite evaluation problem for arbitrary lengths we reduce to the special case of the preceding section by padding the input vector with zeros. Following the approach of pruned and truncated FFT algorithms, we lessen the penalty incurred by having to solve the larger problems by pruning those steps of the algorithm that are specific to the computation of unwanted entries in the output. Thus, we consider the following revised problem in this section: given the coefficients of a polynomial $F \in \mathbb{F}[x]_{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$, compute $(D^{i \operatorname{div} q} F)(\omega_{i \bmod q})$ for

$i \in \{0, \ldots, c-1\}$. The length ℓ Hermite evaluation problem is then captured by taking $n = \lceil \log_2 \lceil \ell / q \rceil \rceil$ and $c = \ell$.

The Hermite evaluation algorithm is described in Algorithm 2. The algorithm operates on a vector $(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ that initially contains the coefficients of a polynomial $F \in \mathbb{F}[x]_{2^n q}$, and overwrites a_i with $(D^{i \operatorname{div} q} F)(\omega_{i \bmod q})$ for i less than the input value c. The remaining entries of the vector are either unchanged or set to intermediate values from the computation. If $c > 2^{n-1} q$, then the algorithm follows the steps described in the preceding section for the length $2^n q$ problem, with the exception that the recursive call used to evaluate $F_1 + D^{2^{n-1}} F_0$ only computes the $c - 2^{n-1} q$ values required for the output. If $c \leq 2^{n-1} q$, then the output depends on F_0 only, so only F_0 is computed (by the function $\mathsf{PrepareLeft}$) and recursively evaluated. Once again, the recursion terminates with $n = 0$, which is handle by an algorithm $\mathsf{Evaluate}$ that satisfies the specifications of Algorithm 1.

Algorithm 1 $\mathsf{Evaluate}((a_0, \ldots, a_{q-1}), c)$

Input: $(a_0, \ldots, a_{q-1}) \in \mathbb{F}^q$ and $c \in \{1, \ldots, q\}$ such that $\sum_{i=0}^{q-1} a_i x^i = F$ for some $F \in \mathbb{F}[x]_q$.
Output: $a_i = F(\omega_i)$ for $i \in \{0, \ldots, c-1\}$.

Algorithm 1 may be realised with a complexity of $\mathcal{O}(\mathsf{M}(q) + \mathsf{M}(c) \log c)$ operations in $\mathbb{F}$ by the use of remainder trees. For small values of c, one can apply Horner's rule for each of the c evaluation points. Naive matrix-vector products are efficient for small q, while additive and (standard) multiplicative FFT algorithms become more efficient for large q. For multiplicative FFT algorithms to be used, the multiplicative group of the field must have smooth cardinality, and it is necessary to first reduce modulo $x^{q-1} - 1$ at the cost of one addition. Moreover, if one has control over the enumeration of the field, then it is possible to obtain a better complexity for $c < q$ by using the truncated FFT algorithm of Larrieu [23].

Proposition 4.1. *Algorithm 2 is correct if Algorithm 1 is correctly implemented.*

Proof. Under the assumption that Algorithm 1 has been correctly implemented, we use induction to show that for all $n \in \mathbb{N}$, Algorithm 2 produces the correct output when given inputs $(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$. Therefore, suppose that Algorithm 1 has been correctly implemented. Then for inputs with $n = 0$, the algorithm trivially produces the correct output since Algorithm 1 is simply applied in this case. Let $n \geq 1$ and suppose that Algorithm 2 functions correctly for all inputs with smaller values of n. Suppose that $(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ are given to the algorithm as inputs, and let $F \in \mathbb{F}[x]_{2^n q}$ be the corresponding polynomial for which the input requirements are satisfied. Let $F_0, F_1 \in \mathbb{F}[x]_{2^{n-1} q}$ such that (3.1) and, equivalently, (3.3) hold.

Suppose that $c > 2^{n-1} q$. Then (3.3) implies that Lines 4 and 5 set a_i equal to the coefficient of x^i in F_0, and $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in F_1, for $i \in \{0, \ldots, 2^{n-1} q - 1\}$. Consequently, Lemma 3.2 implies that Lines 6 to 8 set $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in $F_1 + D^{2^{n-1}} F_0$ for $i \in \{0, \ldots, 2^{n-1} q - 1\}$. As these three lines do not modify $a_0, \ldots, a_{2^{n-1} q - 1}$, which contain the coefficients of F_0, the induction hypothesis and Lemma 3.1 imply that the recursive call of Lines 9 sets

$$a_i = \big(D^{i \operatorname{div} q} F_0\big)(\omega_{i \bmod q}) = \big(D^{i \operatorname{div} q} F\big)(\omega_{i \bmod q}) \tag{4.1}$$

Algorithm 2 HermiteEvaluate$((a_0, \ldots, a_{2^n q-1}), c)$

Input: $(a_0, \ldots, a_{2^n q-1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ such that $n \in \mathbb{N}$ and $\sum_{i=0}^{2^n q-1} a_i x^i = F$ for some $F \in \mathbb{F}[x]_{2^n q}$.
Output: $a_i = (D^{i \operatorname{div} q} F)(\omega_{i \bmod q})$ for $i \in \{0, \ldots, c-1\}$.

1: If $n = 0$:
2: Evaluate$((a_0, \ldots, a_{q-1}), c)$ */* Algorithm 1 */*
3: Else if $c > 2^{n-1}q$:
4: For $i = 2^{n-1}(q+1) - 1, 2^{n-1}(q+1) - 2, \ldots, 2^{n-1}$:
5: $a_i \leftarrow a_i + a_{2^{n-1}(q-1)+i}$
6: For $i = q/2, \ldots, q-1$:
7: For $j = 0, \ldots, 2^{n-1} - 1$:
8: $a_{2^n i+j} \leftarrow a_{2^n i+j} + a_{2^n i+j-(q-1)2^{n-1}}$
9: HermiteEvaluate$((a_0, \ldots, a_{2^{n-1}q-1}), 2^{n-1}q)$
10: HermiteEvaluate$((a_{2^{n-1}q}, \ldots, a_{2^n q-1}), c - 2^{n-1}q)$
11: Else:
12: PrepareLeft$(a, 0)$
13: HermiteEvaluate$((a_0, \ldots, a_{2^{n-1}q-1}), c)$

14: **Function** PrepareLeft$((a_0, \ldots, a_{2^n q-1}), c)$:
15: For $i = \max(c, 2^{n-1}), \ldots, 2^{n-1}q - 1$:
16: $a_i \leftarrow a_i + a_{2^{n-1}(q-1)+i}$
17: For $i = \max(c, 2^{n-1}), \ldots, 2^n - 1$:
18: $a_i \leftarrow a_i + a_{2^n(q-1)+i}$

for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Similarly, the recursive call of Line 10 sets

$$\begin{aligned} a_{2^{n-1}q+i} &= \Big(D^{i \operatorname{div} q}\Big(F_1 + D^{2^{n-1}} F_0\Big)\Big)(\omega_{i \bmod q}) \\ &= \Big(D^{2^{n-1}+(i \operatorname{div} q)} F\Big)(\omega_{i \bmod q}) \\ &= \Big(D^{(2^{n-1}q+i) \operatorname{div} q} F\Big)\big(\omega_{(2^{n-1}q+i) \bmod q}\big) \end{aligned} \tag{4.2}$$

for $i \in \{0, \ldots, c - 2^{n-1}q - 1\}$. The algorithm stops at this point, and thus produces the correct output.

Suppose now that $c \leq 2^{n-1}q$. Then (3.3) implies that Line 12 sets a_i equal to the coefficient of x^i in F_0 for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Consequently, the induction hypothesis and Lemma 3.1 imply that (4.1) holds for $i \in \{0, \ldots, c-1\}$ after the recursive call of Line 13 has been performed. The algorithm stops at this point, and thus produces the correct output. □

For $i, j \in \mathbb{Z}$ such that j is nonzero, define $i \bmod^* j = i - (\lceil i/j \rceil - 1)j$. The following proposition bounds the additive and multiplicative complexities of Algorithm 2 in term of those of Algorithm 1.

Proposition 4.2. *For $n \in \mathbb{N}$, define $A_n, M_n : \{1, \ldots, 2^n q\} \to \mathbb{N}$ as follows: $A_n(c)$ and $M_n(c)$ are respectively the number of additions and multiplications in $\mathbb{F}$ performed by Algorithm 2 (for some implementation of Algorithm 1) when given inputs*

$(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ *and* $c \in \{1, \ldots, 2^n q\}$. *Then*

$$A_n(c) \leq A_0(q)(\lceil c/q \rceil - 1) + A_0(c \bmod^* q) + \left(\frac{3}{4}\lceil \log_2 \lceil c/q \rceil \rceil - \frac{1}{4}\right)(\lceil c/q \rceil - 1)q + (2^n - 1)q$$

and $M_n(c) = M_0(q)(\lceil c/q \rceil - 1) + M_0(c \bmod^* q)$ *for* $n \in \mathbb{N}$ *and* $c \in \{1, \ldots, 2^n q\}$.

Proof. For nonzero $n \in \mathbb{N}$, define the indicator function $\delta_n : \{1, \ldots, 2^n q\} \to \{0, 1\}$ by $\delta_n(c) = 1$ if and only if $c > 2^{n-1}q$. Then given inputs $(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ for some nonzero $n \in \mathbb{N}$, Lines 4 to 8 of Algorithm 2 perform $\delta_n(c)(3/4)2^n q$ additions, Line 9 performs $\delta_n(c) A_{n-1}(2^{n-1}q)$ additions and $\delta_n(c) M_{n-1}(2^{n-1}q)$ multiplications, and Line 10 performs $\delta_n(c) A_{n-1}(c - \delta_n(c)2^{n-1}q)$ additions and $\delta_n(c) M_{n-1}(c - \delta_n(c)2^{n-1}q)$ multiplications. Furthermore, Line 12 performs $(1 - \delta_n(c))2^{n-1}q$ additions, and Line 13 performs $(1 - \delta_n(c)) A_{n-1}(c - \delta_n(c)2^{n-1}q)$ additions and $(1 - \delta_n(c)) M_{n-1}(c - \delta_n(c)2^{n-1}q)$ multiplications. Summing these contributions, it follows that

$$A_n(c) = A_{n-1}\big(c - \delta_n(c)2^{n-1}q\big) + 2^{n-1}q + \delta_n(c)\big(A_{n-1}\big(2^{n-1}q\big) + 2^{n-2}q\big) \tag{4.3}$$

and

$$M_n(c) = M_{n-1}\big(c - \delta_n(c)2^{n-1}q\big) + \delta_n(c) M_{n-1}\big(2^{n-1}q\big) \tag{4.4}$$

for nonzero $n \in \mathbb{N}$ and $c \in \{1, \ldots, 2^n q\}$. In particular,

$$A_n(2^n q) = 2A_{n-1}\big(2^{n-1}q\big) + \frac{3}{4}2^n q \quad \text{and} \quad M_n(2^n q) = 2M_{n-1}\big(2^{n-1}q\big) \tag{4.5}$$

for nonzero $n \in \mathbb{N}$. Thus,

$$A_n(2^n q) = 2^n\left(A_0(q) + \frac{3}{4}nq\right) \quad \text{and} \quad M_n(2^n q) = 2^n M_0(q) \quad \text{for } n \in \mathbb{N}. \tag{4.6}$$

Substituting these equations into (4.3) and (4.4), it follows that

$$A_n(c) = A_{n-1}\big(c - \delta_n(c)2^{n-1}q\big) + 2^{n-1}q + \delta_n(c)2^{n-1}\left(A_0(q) + \frac{3}{4}(n-1)q + \frac{q}{2}\right) \tag{4.7}$$

and

$$M_n(c) = M_{n-1}\big(c - \delta_n(c)2^{n-1}q\big) + \delta_n(c)2^{n-1} M_0(q) \tag{4.8}$$

for nonzero $n \in \mathbb{N}$ and $c \in \{1, \ldots, 2^n q\}$.

For $n, c, \delta \in \mathbb{N}$ such that n is nonzero, we have $\lceil (c - \delta 2^{n-1}q)/q \rceil = \lceil c/q \rceil - \delta 2^{n-1}$ and $c - \delta 2^{n-1}q \bmod^* q = c \bmod^* q$. Consequently, the formula for $M_n(c)$ stated in the proposition follows from (4.8) by induction on n. If $n \in \mathbb{N}$ is nonzero, $c \in \{1, \ldots, 2^n q\}$ and

$$\lceil c/q \rceil - 1 = i_0 + i_1 \cdot 2 + \cdots + i_{n-1} \cdot 2^{n-1}, \tag{4.9}$$

with $i_0, \ldots, i_{n-1} \in \{0, 1\}$, then $i_{n-1} = \delta_n(c)$. Therefore, it follows from (4.7) by induction on n, that

$$A_n(c) = A_0(c \bmod^* q) + (2^n - 1)q + \left(A_0(q) + \frac{q}{2}\right)(\lceil c/q \rceil - 1) + \frac{3}{4}q \sum_{k=0}^{n-1} 2^k i_k k \tag{4.10}$$

for $n \in \mathbb{N}$ and $c \in \{1, \ldots, 2^n q\}$, where $i_0, \ldots, i_{n-1} \in \{0, 1\}$ are the coefficients of the binary expansion (4.9). Here,

$$\sum_{k=0}^{n-1} 2^k i_k k \leq \max(\lceil \log_2 \lceil c/q \rceil \rceil - 1, 0) \sum_{k=0}^{n-1} 2^k i_k = (\lceil \log_2 \lceil c/q \rceil \rceil - 1)(\lceil c/q \rceil - 1),$$

since $i_k = 0$ if $k \geq \lceil \log_2 \lceil c/q \rceil \rceil$. Combining this inequality with (4.10) yields the upper bound on $A_n(c)$ stated in the proposition. □

The functions A_0 and M_0 defined in Proposition 4.2 describe the additive and multiplicative complexities of the implementation of Algorithm 1. When $n = 0$ or $c > 2^{n-1}q$ for some nonzero n, as may be assumed when solving the Hermite evaluation problem, the third and fourth terms of the bound on $A_n(c)$ are in $\mathcal{O}(c \log \lceil c/q \rceil)$. By taking A_0 and M_0 to be in $\mathcal{O}(\mathsf{M}(q) + \mathsf{M}(c) \log c)$, and making the common assumption (used, for instance, in [41]) that $\mathsf{M}(\ell)/\ell$ is a nondecreasing function of ℓ, it follows that the length ℓ Hermite evaluation problem can be solved with $\mathcal{O}(\mathsf{M}(\ell) \log q + \ell \log \lceil \ell/q \rceil)$ operations in $\mathbb{F}$ by Algorithm 2. For this application, the number of additions performed by Algorithm 2 may be reduced by adapting the algorithm to take into account the zeros that initially occupy the $2^n q - \ell$ rightmost entries of the vector $(a_0, \ldots, a_{2^n q - 1})$.

5. Interpolation algorithm

To solve the Hermite interpolation problem for arbitrary lengths, we use an approach analogous to that employed by the inverse truncated FFT algorithm of Larrieu [23] by reducing to a length $2^n q$ problem under the assumption that the new entries of the output that result from extending the problem are provided as inputs. Thus, we consider the following problem in this section: given $c \in \{1, \ldots, 2^n q\}$ and $(h_0, \ldots, h_{c-1}, f_c, \ldots, f_{2^n q - 1}) \in \mathbb{F}^{2^n q}$, compute $f_0, \ldots, f_{c-1} \in \mathbb{F}$ such that $F = \sum_{i=0}^{2^n q - 1} f_i x^i$ satisfies $(D^{i \operatorname{div} q} F)(\omega_{i \bmod q}) = h_i$ for $i \in \{0, \ldots, c-1\}$. Here, existence and uniqueness of $f_0, \ldots, f_{c-1}$ follow readily from Lemma 2.2. The length ℓ Hermite interpolation problem is then captured as an instance of the new problem by taking $n = \lceil \log_2 \lceil \ell/q \rceil \rceil$, $c = \ell$ and $f_c = \cdots = f_{2^n q - 1} = 0$.

The Hermite interpolation algorithm is described in Algorithm 4. If $c > 2^{n-1}q$, then the algorithm closely follows the approach described in Section 3 by recursively computing the polynomials F_0 and $F_1 + D^{2^{n-1}} F_0$, before using Lemma 3.2 and the expansion (3.3) to compute the desired coefficients of F. The recursive call used to recover $F_1 + D^{2^{n-1}} F_0$ cannot be made without first computing the coefficient of x^i in the polynomial for $i \geq c - 2^{n-1}q$. Consequently, after the algorithm has recovered F_0, the required coefficients are computed by function the PrepareRight, which steps through Lines 4 to 8 of Algorithm 2 while only modifying those entries a_i with indices $i \geq c$. If $c \leq 2^{n-1}q$, then the function PrepareLeft from Algorithm 2 is used to recover the coefficient of x^i in F_0 for $i \geq c$ before the remaining coefficients of the polynomial are recursively computed. The function PrepareLeft, which is its own inverse for fixed c, is then used to compute the lower order coefficients of the output. The base case of the recursion is handled by an algorithm Interpolate that satisfies the specifications of Algorithm 3.

Algorithm 3 may be realised with a complexity of $\mathcal{O}(\mathsf{M}(q) + \mathsf{M}(c) \log c)$ operations in $\mathbb{F}$ by the use of a fast Chinese remainder algorithm: $\sum_{i=0}^{c-1} f_i x^i$ is equal to the sum of the polynomial $C \in \mathbb{F}[x]_c$ that satisfies $C(\omega_i) = F(\omega_i)$ for $i \in \{0, \ldots, c-1\}$,

Algorithm 3 $\mathsf{Interpolate}((a_0, \ldots, a_{q-1}), c)$

Input: $(a_0, \ldots, a_{q-1}) \in \mathbb{F}^q$ and $c \in \{1, \ldots, q\}$ such that for some polynomial $F = \sum_{i=0}^{q-1} f_i x^i \in \mathbb{F}[x]_q$ the following conditions hold:
(1) $a_i = F(\omega_i)$ for $i \in \{0, \ldots, c-1\}$, and
(2) $a_i = f_i$ for $i \in \{c, \ldots, q-1\}$.
Output: $a_i = f_i$ for $i \in \{0, \ldots, q-1\}$.

and the remainder of $\sum_{i=c}^{q-1} f_i x^i$ upon division by $\prod_{i=0}^{c-1}(x - \omega_i)$, with the product being computed as part of the Chinese remainder algorithm. If the enumeration of the field may be chosen freely and its multiplicative group has smooth cardinality, then a better complexity is obtained by using the inverse truncated FFT algorithm of Larrieu [23]. In doing so, one should set $\omega_{q-1} = 0$ so that only a single addition is required on top of the call to the FFT algorithm, since $F(\omega) = f_{q-1} + \sum_{i=0}^{q-2} f_i \omega^i$ for nonzero $\omega \in \mathbb{F}$.

Proposition 5.1. *Algorithm 4 is correct if Algorithm 3 is correctly implemented.*

Proof. Under the assumption that Algorithm 3 has been correctly implemented, we use induction to show that for all $n \in \mathbb{N}$, Algorithm 4 produces the correct output when given inputs $(a_0, \ldots, a_{2^n q-1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$. Therefore, suppose that Algorithm 3 has been correctly implemented. Then for inputs with $n = 0$, the algorithm trivially produces the correct output since Algorithm 3 is simply applied in this case. Let $n \geq 1$ and suppose that Algorithm 4 functions correctly for all inputs with smaller values of n. Suppose that $(a_0, \ldots, a_{2^n q-1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ are given to the algorithm as inputs, and let $F \in \mathbb{F}[x]_{2^n q}$ be the corresponding polynomial for which the input requirements are satisfied. Let $F_0, F_1 \in \mathbb{F}[x]_{2^{n-1} q}$ such that (3.1) and, equivalently, (3.3) hold.

Suppose that $c > 2^{n-1} q$. Then Lemma 3.1 implies that (4.1) initially holds for $i \in \{0, \ldots, 2^{n-1} q - 1\}$. Consequently, the induction hypothesis and Lemma 2.2 imply that the recursive call of Line 4 sets a_i equal to the coefficient of x^i in F_0 for $i \in \{0, \ldots, 2^{n-1} q - 1\}$. Thus, when the function $\mathsf{PrepareRight}$ is called in Line 5, (3.3) implies that Lines 17 and 18 of the function set $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in F_1 for $i \in \{c - 2^{n-1} q, \ldots, 2^{n-1} q - 1\}$. Then Lemma 3.2 implies that Lines 19 to 25 of the function set $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in $F_1 + D^{2^{n-1}} F_0$ for $i \in \{c - 2^{n-1} q, \ldots, 2^{n-1} q - 1\}$. The entries $a_{2^{n-1} q}, \ldots, a_{c-1}$ are so far unchanged by the algorithm. Thus, Lemma 3.1 implies that (4.2) holds for $i \in \{0, \ldots, c - 2^{n-1} q - 1\}$. The induction hypothesis and Lemma 2.2 therefore imply that the recursive call of Line 6 sets $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in $F_1 + D^{2^{n-1}} F_0$ for $i \in \{0, \ldots, c - 2^{n-1} q - 1\}$. Hence, after the recursive call, the left half of the vector $(a_0, \ldots, a_{2^n q-1})$ contains the coefficients of F_0, while its right half contains the coefficients of $F_1 + D^{2^{n-1}} F_0$. Consequently, Lemma 3.2 implies that Lines 7 to 9 set $a_{2^{n-1} q + i}$ equal to the coefficient of x^i in F_1 for $i \in \{0, \ldots, 2^{n-1} q - 1\}$, then (3.3) implies that Lines 10 to 11 set a_i equal to the coefficient of x^i in F for $i \in \{0, \ldots, 2^n q - 1\}$. The algorithm stops at this point, and thus produces the correct output.

Suppose now that $c \leq 2^{n-1} q$. Then (3.3) implies that the call to $\mathsf{PrepareLeft}$ in Line 13 sets a_i equal to the coefficient of x^i in F_0 for $i \in \{c, \ldots, 2^{n-1} q - 1\}$. This call to $\mathsf{PrepareLeft}$ does not modify $a_0, \ldots, a_{c-1}$. Thus, Lemma 3.1 implies

Algorithm 4 HermiteInterpolate$((a_0, \ldots, a_{2^n q-1}), c)$

Input: $(a_0, \ldots, a_{2^n q-1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ such that $n \in \mathbb{N}$ and for some polynomial $F = \sum_{i=0}^{2^n q-1} f_i x^i \in \mathbb{F}[x]_{2^n q}$ the following conditions hold:
(1) $a_i = (D^{i \operatorname{div} q} F)(\omega_{i \bmod q})$ for $i \in \{0, \ldots, c-1\}$, and
(2) $a_i = f_i$ for $i \in \{c, \ldots, 2^n q - 1\}$.
Output: $a_i = f_i$ for $i \in \{0, \ldots, 2^n q - 1\}$.

1: If $n = 0$:
2: HermiteInterpolate$((a_0, \ldots, a_{q-1}), c)$ *(* Algorithm 3 *)*
3: Else if $c > 2^{n-1}q$:
4: HermiteInterpolate$((a_0, \ldots, a_{2^{n-1}q-1}), 2^{n-1}q)$
5: PrepareRight$((a_0, \ldots, a_{2^n q-1}), c)$
6: HermiteInterpolate$((a_{2^{n-1}q}, \ldots, a_{2^n q-1}), c - 2^{n-1}q)$
7: For $i = q/2, \ldots, q-1$:
8: For $j = 0, \ldots, 2^{n-1} - 1$:
9: $a_{2^n i+j} \leftarrow a_{2^n i+j} + a_{2^n i+j-(q-1)2^{n-1}}$
10: For $i = 2^{n-1}, \ldots, 2^{n-1}(q+1) - 1$:
11: $a_i \leftarrow a_i + a_{2^{n-1}(q-1)+i}$
12: Else:
13: PrepareLeft(a, c) *(* From Algorithm 2 *)*
14: HermiteInterpolate$((a_0, \ldots, a_{2^{n-1}q-1}), c)$
15: PrepareLeft$(a, 0)$

16: **Function** PrepareRight$((a_0, \ldots, a_{2^n q-1}), c)$:
17: For $i = 2^{n-1}(q+1) - 1, 2^{n-1}(q+1) - 2, \ldots, c$:
18: $a_i \leftarrow a_i + a_{2^{n-1}(q-1)+i}$
19: $t \leftarrow c \operatorname{div} 2^n$, $r \leftarrow \min(c \bmod 2^n, 2^{n-1})$
20: For $j = 0, \ldots, r-1$:
21: For $i = t+1, \ldots, q-1$:
22: $a_{2^n i+j} \leftarrow a_{2^n i+j} + a_{2^n i+j-(q-1)2^{n-1}}$
23: For $j = r, \ldots, 2^{n-1} - 1$:
24: For $i = t, \ldots, q-1$:
25: $a_{2^n i+j} \leftarrow a_{2^n i+j} + a_{2^n i+j-(q-1)2^{n-1}}$

that (4.1) holds for $i \in \{0, \ldots, c-1\}$ when the recursive call of Line 14 is made. The induction hypothesis and Lemma 2.2 therefore imply that Line 14 sets a_i equal to the coefficient of x^i in F_0 for $i \in \{0, \ldots, c-1\}$. Hence, after the recursive call, the left half of the vector $(a_0, \ldots, a_{2^n q-1})$ contains the coefficients of F_0, while the entries in the right half still retain their initial values, with a_i equal to the coefficient of x^i in F for $i \in \{2^{n-1}q, \ldots, 2^n q - 1\}$. Thus, (3.3) implies that the call to PrepareLeft in Line 15 sets a_i equal to the coefficient of x^i in F for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. The algorithm stops at this point, and thus produces the correct output. □

Proposition 5.2. *For $n \in \mathbb{N}$, define $A_n, M_n : \{1, \ldots, 2^n q\} \to \mathbb{N}$ as follows: $A_n(c)$ and $M_n(c)$ are respectively the number of additions and multiplications in $\mathbb{F}$ performed by Algorithm 4 (for some implementation of Algorithm 3) when given inputs*

$(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ *and* $c \in \{1, \ldots, 2^n q\}$. *Then*

$$A_n(c) \leq A_0(q)(\lceil c/q \rceil - 1) + A_0(c \bmod^* q) + \left(\frac{7}{4}\lceil \log_2 \lceil c/q \rceil \rceil - n - \frac{3}{4}\right)(\lceil c/q \rceil - 1)q + (2^n - 1)(2q + 1)$$

and $M_n(c) = M_0(q)(\lceil c/q \rceil - 1) + M_0(c \bmod^* q)$ *for* $n \in \mathbb{N}$ *and* $c \in \{1, \ldots, 2^n q\}$.

Proof. The proof of the bound on $A_n(c)$ follows along similar lines to that of Proposition 4.2, but with the addition of having to bound the number of additions performed in Lines 5 and 13 of the algorithm. As no multiplications are performed by either of these lines, they may be ignored when proving the formula for $M_n(c)$. In doing so, the proof follows along identical lines to that of Proposition 4.2, and is therefore omitted.

Suppose that Algorithm 4 has been given inputs $(a_0, \ldots, a_{2^n q - 1}) \in \mathbb{F}^{2^n q}$ and $c \in \{1, \ldots, 2^n q\}$ for some nonzero $n \in \mathbb{N}$. Let $t = c \operatorname{div} 2^n$ and $r = \min(c \bmod 2^n, 2^{n-1})$, as defined in Line 19 of the function PrepareRight. If $c > 2^{n-1} q$, then the call to PrepareRight in Line 5 of the Algorithm 4 performs

$$\max(2^{n-1}(q+1) - c, 0) + \left(2^{n-1}(q-t) - r\right) < 2^{n-1} + \left(2^{n-1} q - c/2\right)$$

additions. In particular, no additions are performed if $c = 2^n q$. Consequently, A_n once again satisfies the recurrence (4.5) for nonzero $n \in \mathbb{N}$, and, as a result, also satisfies (4.6). If $c \leq 2^{n-1} q$, then Line 13 of Algorithm 4 performs at most $(2^{n-1} q - c) + (2^n - 2^{n-1})$ additions. By summing the contributions of each line of Algorithm 4 in the manner of the proof of Proposition 4.2, with the two bounds used for the contributions of Lines 5 and 13, it follows that

$$A_n(c) \leq A_{n-1}\left(c - \delta_n(c) 2^{n-1} q\right) + 2^{n-1}(2q+1) + \delta_n(c) 2^{n-1}\left(A_0(q) + \frac{3}{4}(n-1)q + \frac{q}{2}\right) - \left(1 - \frac{\delta_n(c)}{2}\right)c$$

for nonzero $n \in \mathbb{N}$ and $c \in \{1, \ldots, 2^n q\}$, where δ_n is the indicator function defined in the proof of Proposition 4.2. Therefore,

$$A_n(c) \leq A_0(q)(\lceil c/q \rceil - 1) + A_0(c \bmod^* q) + (2^n - 1)(2q+1) + \left(\frac{3}{4}\lceil \log_2 \lceil c/q \rceil \rceil - \frac{1}{4}\right)(\lceil c/q \rceil - 1)q - \sum_{k=0}^{n-1}\left(1 - \frac{i_k}{2}\right)\left(c - q \sum_{j=k+1}^{n-1} 2^j i_j\right)$$

for $n \in \mathbb{N}$ and $c \in \{1, \ldots, 2^n q\}$, where $i_0, \ldots, i_{n-1} \in \{0, 1\}$ are the coefficients of the binary expansion (4.9). The upper bound on $A_n(c)$ stated in the proposition is then obtained by observing that

$$\begin{aligned}\sum_{k=0}^{n-1}\left(1 - \frac{i_k}{2}\right)\left(c - q \sum_{j=k+1}^{n-1} 2^j i_j\right) &\geq \sum_{k=\lceil \log_2 \lceil c/q \rceil \rceil}^{n-1} c + \sum_{k=\max(\lceil \log_2 \lceil c/q \rceil \rceil - 1, 0)}^{\lceil \log_2 \lceil c/q \rceil \rceil - 1} \frac{c}{2} \\ &\geq \left(n - \lceil \log_2 \lceil c/q \rceil \rceil + \frac{1}{2}\right)(\lceil c/q \rceil - 1)q\end{aligned}$$

for $n \in \mathbb{N}$, $c \in \{1, \ldots, 2^n q\}$ and $i_0, \ldots, i_{n-1} \in \{0, 1\}$ such that (4.9) holds. □

By taking A_0 and M_0 to be in $\mathcal{O}(\mathsf{M}(q) + \mathsf{M}(c) \log c)$, it follows from Proposition 5.2 that the length ℓ Hermite interpolation problem can be solved with

$\mathcal{O}(\mathsf{M}(\ell)\log q + \ell\log\lceil \ell/q\rceil)$ operations in $\mathbb{F}$ by Algorithm 4. The number of additions performed by the algorithm in this setting may once again be reduced by taking into account the zeros that initially occupy the $2^n q - \ell$ rightmost entries of the vector $(a_0, \ldots, a_{2^n q-1})$. Moreover, as some of these entries are changed during the course of the algorithm, but ultimately are equal to zero again at its end, it is possible to save further additions by not performing those steps specific to restoring the entries to zero.

References

1. A. V. Aho, K. Steiglitz, and J. D. Ullman, *Evaluating polynomials at fixed sets of points*, SIAM J. Comput. **4** (1975), no. 4, 533–539.
2. Daniel Augot, Françoise Levy-dit-Vehel, and Abdullatif Shikfa, *A storage-efficient and robust private information retrieval scheme allowing few servers*, Cryptology and network security, Lecture Notes in Comput. Sci., vol. 8813, Springer, Cham, 2014, pp. 222–239.
3. Daniel J. Bernstein, *Scaled remainder trees*, Available from https://cr.yp.to/arith/scaledmod-20040820.pdf, 2004.
4. Daniel J. Bernstein and Tung Chou, *Faster binary-field multiplication and faster binary-field MACs*, Selected areas in cryptography—SAC 2014, Lecture Notes in Comput. Sci., vol. 8781, Springer, Cham, 2014, pp. 92–111.
5. Daniel J. Bernstein, Tung Chou, and Peter Schwabe, *McBits: Fast constant-time code-based cryptography*, Cryptographic Hardware and Embedded Systems - CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. (Berlin, Heidelberg) (Guido Bertoni and Jean-Sébastien Coron, eds.), Springer Berlin Heidelberg, 2013, pp. 250–272.
6. Dario Bini and Victor Y. Pan, *Polynomial and matrix computations, vol. 1: Fundamental algorithms*, Progress in Theoretical Computer Science, Birkhäuser Boston, Inc., Boston, MA, 1994.
7. A. Borodin and R. Moenck, *Fast modular transforms*, J. Comput. System Sci. **8** (1974), 366–386.
8. A. Bostan, G. Lecerf, B. Salvy, É. Schost, and B. Wiebelt, *Complexity issues in bivariate polynomial factorization*, ISSAC 2004, ACM, New York, 2004, pp. 42–49.
9. A. Bostan, G. Lecerf, and É. Schost, *Tellegen's principle into practice*, Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '03, ACM, 2003, pp. 37–44.
10. Alin Bostan and Éric Schost, *Polynomial evaluation and interpolation on special sets of points*, J. Complexity **21** (2005), no. 4, 420–446.
11. David G. Cantor, *On arithmetical algorithms over finite fields*, J. Combin. Theory Ser. A **50** (1989), no. 2, 285–300.
12. David G. Cantor and Erich Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta Inform. **28** (1991), no. 7, 693–701.
13. Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang, *Faster multiplication for long binary polynomials*, 2017, `arXiv:1708.09746 [cs.SC]`.
14. Francis Y. Chin, *A generalized asymptotic upper bound on fast polynomial evaluation and interpolation*, SIAM J. Comput. **5** (1976), no. 4, 682–690.
15. James W. Cooley and John W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp. **19** (1965), 297–301.
16. Nicholas Coxon, *Fast systematic encoding of multiplicity codes*, `arXiv:1704.07083 [cs.IT]`, 2017.
17. Charles M. Fiduccia, *Polynomial evaluation via the division algorithm the fast fourier transform revisited*, Proceedings of the Fourth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '72, ACM, 1972, pp. 88–93.
18. N. J. Fine, *Binomial coefficients modulo a prime*, Amer. Math. Monthly **54** (1947), 589–592.
19. Shuhong Gao and Todd Mateer, *Additive fast Fourier transforms over finite fields*, IEEE Trans. Inform. Theory **56** (2010), no. 12, 6265–6272.
20. David Harvey, *A cache-friendly truncated FFT*, Theoret. Comput. Sci. **410** (2009), no. 27-29, 2649–2658.

21. David Harvey and Daniel S. Roche, *An in-place truncated Fourier transform and applications to polynomial multiplication*, ISSAC 2010—Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2010, pp. 325–329.
22. Swastik Kopparty, *Some remarks on multiplicity codes*, Discrete geometry and algebraic combinatorics, Contemp. Math., vol. 625, Amer. Math. Soc., Providence, RI, 2014, pp. 155–176.
23. Robin Larrieu, *The truncated Fourier transform for mixed radices*, Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC'17, ACM, 2017, pp. 261–268.
24. Sian-Jheng Lin, Tareq Y. Al-Naffouri, and Yunghsiang S. Han, *FFT algorithm for binary extension finite fields and its application to Reed-Solomon codes*, IEEE Trans. Inform. Theory **62** (2016), no. 10, 5343–5358.
25. Sian-Jheng Lin, Tareq Y. Al-Naffouri, Yunghsiang S. Han, and Wei-Ho Chung, *Novel polynomial basis with fast Fourier transform and its application to Reed-Solomon erasure codes*, IEEE Trans. Inform. Theory **62** (2016), no. 11, 6284–6299.
26. Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han, *Novel polynomial basis and its application to Reed-Solomon erasure codes*, 55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014, IEEE Computer Soc., Los Alamitos, CA, 2014, pp. 316–325.
27. Edouard Lucas, *Théorie des Fonctions Numériques Simplement Périodiques. [Continued]*, Amer. J. Math. **1** (1878), no. 3, 197–240.
28. J. Markel, *FFT pruning*, IEEE Transactions on Audio and Electroacoustics **19** (1971), no. 4, 305–311.
29. Todd Mateer, *Fast Fourier Transform algorithms with applications*, Ph.D. thesis, Clemson University, August 2008.
30. R. Moenck and A. Borodin, *Fast modular transforms via division*, Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972) (Washington, DC, USA), SWAT '72, IEEE Computer Society, 1972, pp. 90–96.
31. Vadim Olshevsky and Amin Shokrollahi, *Matrix-vector product for confluent Cauchy-like matrices with application to confluent rational interpolation*, Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 573–581.
32. Victor Y. Pan, *Structured matrices and polynomials: unified superfast algorithms*, Birkhäuser Boston, Inc., Boston, MA; Springer-Verlag, New York, 2001.
33. A. Schönhage, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2*, Acta Informat. **7** (1976/77), no. 4, 395–398. MR 0436663
34. A. Schönhage and V. Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing (Arch. Elektron. Rechnen) **7** (1971), 281–292.
35. H. V. Sorensen and C. S. Burrus, *Efficient computation of the DFT with only a subset of input or output points*, IEEE Transactions on Signal Processing **41** (1993), no. 3, 1184–1200.
36. Joris van der Hoeven, *The truncated Fourier transform and applications*, ISSAC 2004, ACM, New York, 2004, pp. 290–296.
37. Joris van der Hoeven, *Faster Chinese remaindering*, Tech. report, HAL, 2016, `http://hal.archives-ouvertes.fr/hal-01403810`.
38. T. M. Vari, *Some complexity results for a class of Toeplitz matrices*, Tech. report, Dept. of Computer Sci. and Math., York Univ., Toronto, 1974.
39. Joachim von zur Gathen, *Functional decomposition of polynomials: the tame case*, J. Symbolic Comput. **9** (1990), no. 3, 281–299.
40. Joachim von zur Gathen and Jürgen Gerhard, *Fast algorithms for Taylor shifts and certain difference equations*, Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1997, pp. 40–47.
41. Joachim von zur Gathen and Jürgen Gerhard, *Modern computer algebra*, third ed., Cambridge University Press, Cambridge, 2013.
42. David Woodruff and Sergey Yekhanin, *A geometric approach to information-theoretic private information retrieval*, SIAM J. Comput. **37** (2007), no. 4, 1046–1056.
43. Liyasi Wu, *Revisiting the multiplicity codes: A new class of high-rate locally correctable codes*, 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), September 2015, pp. 509–513.