Fast Hermite interpolation and evaluation over finite fields of characteristic two

Nicholas Coxon

INRIA and Laboratoire d'Informatique de l'École polytechnique, Palaiseau, France.

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.

Keywords: Hermite interpolation, Hermite evaluation, multiplicity codes

1. Introduction

Hermite interpolation is the problem of computing the coefficients of a polynomial given the values of its derivatives up to sufficiently large orders 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 greater than or equal to p. Consequently, it is usual to consider Hermite interpolation and evaluation with respect to the Hasse derivative over such fields when the characteristic is small.

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 $\omega_0, \ldots, \omega_{n-1} \in \mathbb{F}$ and positive integer multiplicities $\ell_0, \ldots, \ell_{n-1}$, the Hermite interpolation problem over $\mathbb{F}$ asks that we compute the coefficients of a polynomial $f \in \mathbb{F}[x]$ of degree strictly less than $\ell = \ell_0 + \cdots + \ell_{n-1}$ when given $(D^i f)(\omega_j)$ for $j \in \{0, \ldots, \ell_i - 1\}$ and $i \in \{0, \ldots, n - 1\}$. 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. This paper considers the problems with respect to the monomial basis $\{x^i \mid i \in \mathbb{N}\}$ of $\mathbb{F}[x]$ only.

In this paper, the complexity of algorithms is measured by counting the number of field operations they perform. Let $\mathsf{M}(\ell)$ denote the number of operations in $\mathbb{F}$ required to multiply two polynomials in $\mathbb{F}[x]$ of degree strictly less than ℓ. Then $\mathsf{M}(\ell)$ may be taken to be in $O(\ell(\log \ell) \log\log \ell)$ (Schönhage and Strassen, 1971; Schönhage, 1976/77; Cantor and Kaltofen, 1991), and may be taken to be in $O(\ell(\log \ell)4^{\log^* \ell})$, where $\log^*$ denotes the iterated logarithm, if the field is finite (Harvey and van der Hoeven, 2017; Harvey et al., 2016, 2017). Throughout

the paper, the common assumption is made (used, for instance, by von zur Gathen and Gerhard (2013)) that $\mathsf{M}(\ell)/\ell$ is an increasing function of ℓ.

The boundary case $\ell_0 = \cdots = \ell_{n-1} = 1$ of the Hermite interpolation and evaluation problems corresponds to standard multipoint interpolation and evaluation, allowing the problems to be solved with $O(\mathsf{M}(\ell)\log\ell)$ operations by the use of remainder trees and fast Chinese remainder algorithms (Fiduccia, 1972; Moenck and Borodin, 1972; Borodin and Moenck, 1974; Bostan et al., 2003, 2004; Bernstein, 2004; see also von zur Gathen and Gerhard, 2013, Chapter 10). If the field admits a suitable "inborn" fast Fourier transform (FFT), as occurs when it is finite, and the evaluation points are fixed, then the algorithms of van der Hoeven (2016) allow a factor of size $O(\log\log\ell)$ to be removed from these estimates. If the evaluation points form a geometric progression, then the complexity of solving the standard interpolation and evaluation problems reduces to $O(\mathsf{M}(\ell))$ operations (Bostan and Schost, 2005). Similarly, the cost of solving both problems reduces to $O(\ell\log\ell)$ operations when the evaluation points coincide with those of a truncated Fourier transform (van der Hoeven, 2004, 2005; Harvey, 2009; Harvey and Roche, 2010; see also Larrieu, 2017).

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

$$f = \sum_{i\in\mathbb{N}} (D^i f)(\omega)(x-\omega)^i \quad \text{for } f \in \mathbb{F}[x] \text{ and } \omega \in \mathbb{F}. \tag{1}$$

Consequently, Hermite interpolation and evaluation at a single evaluation point can be performed with $O(\mathsf{M}(\ell)\log\ell)$ operations in general (Borodin and Moenck, 1974; von zur Gathen, 1990; von zur Gathen and Gerhard, 1997), $O(\mathsf{M}(\ell))$ operations if $(\ell-1)!$ is invertible in the field (Aho et al., 1975; Vari, 1974) (see also von zur Gathen and Gerhard, 1997; Bini and Pan, 1994), and $O(\ell\log\ell)$ operations if the field has characteristic equal to two (Gao and Mateer, 2010).

The first quasi-linear time algorithms for solving the general Hermite problems were proposed by Chin (1976). Truncating the Taylor expansion (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 $O(\mathsf{M}(\ell)\log\ell)$ operations (Chin, 1976; Olshevsky and Shokrollahi, 2000) (see also Bini and Pan, 1994; Pan, 2001).

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 (Coxon, 2018a), encoding and decoding algorithms for multiplicity codes (Kopparty, 2014; Coxon, 2018a) and the codes of Wu (2015), and private information retrieval protocols based on these codes (Woodruff and Yekhanin, 2007; Augot et al., 2014).

When ℓ is a multiple of the order q of the field, 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 $O(\ell\log\ell/q)$ additions. Similarly, the Hermite

evaluation algorithm in this case performs $O(\ell \log \ell/q)$ additions, followed by ℓ/q standard evaluations over the q points. As the multiplicative group of the field is cyclic, minor modifications to the algorithms of Bostan and Schost (2005) allow these standard interpolations and evaluations to be performed with $O(\mathsf{M}(q))$ operations. Consequently, our algorithms perform $O((\ell/q)\mathsf{M}(q) + \ell \log \ell/q)$ operations in this special case.

When ℓ is not a multiple of q, the Hermite interpolation and evaluation algorithms still perform $O(\ell \log\lceil \ell/q \rceil)$ additions and $\lceil \ell/q \rceil - 1$ standard interpolations or evaluations over the q evaluation points. However, each algorithm must also solve one instance of a slightly generalised version of the corresponding standard problem. The generalised evaluation problem may be solved with $O(\mathsf{M}(q))$ operations, while the generalised interpolation problem may be solved with $O(\mathsf{M}(q) \log q)$ operations in general, or $O(\mathsf{M}(q))$ operations if we are free to order the evaluation points and assign their multiplicities. It follows that the Hermite evaluation performs $O(\lceil \ell/q \rceil \mathsf{M}(q) + \ell \log\lceil \ell/q \rceil)$ operations, while the Hermite interpolation algorithms performs $O((\lceil \ell/q \rceil + \log q)\mathsf{M}(q) + \ell \log\lceil \ell/q \rceil)$ operations in general, and $O(\lceil \ell/q \rceil \mathsf{M}(q) + \ell \log\lceil \ell/q \rceil)$ operations in the special case. In particular, $O(\ell \log \ell)$ additions and $O(\ell)$ multiplications are performed by the algorithms when the field is fixed.

The reduction from Hermite to standard problems is introduced 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 (Markel, 1971; Sorensen and Burrus, 1993; van der Hoeven, 2004, 2005; Harvey, 2009; Harvey and Roche, 2010; Larrieu, 2017), 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 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* $j \in \{0, \ldots, i-1\}$ *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 1 follow readily from the definition of Hasse derivatives provided in Section 1. 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 strictly less than ℓ. Then existence and uniqueness for the general Hermite interpolation problem is provided by the following lemma.

Lemma 2. *Let $\omega_0, \ldots, \omega_{n-1} \in \mathbb{F}$ be distinct, $\ell_0, \ldots, \ell_{n-1} \in \mathbb{N}$ be positive, and $\ell = \ell_0 + \cdots + \ell_{n-1}$. Then given elements $h_{i,j} \in \mathbb{F}$ for $i \in \{0, \ldots, \ell_j - 1\}$ and $j \in \{0, \ldots, n-1\}$, there exists a unique polynomial $f \in \mathbb{F}[x]_\ell$ such that $(D^i f)(\omega_j) = h_{i,j}$ for $i \in \{0, \ldots, \ell_j - 1\}$ and $j \in \{0, \ldots, n-1\}$.*

Lemma 2 follows from Property 3 of Lemma 1, which implies that the kernel of the linear map from $\mathbb{F}[x]_\ell$ to $\mathbb{F}^\ell$ given by $f \mapsto ((D^i f)(\omega_j))_{0 \le i < \ell_j, 0 \le j < n}$ 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 maps $c_i, e_i : \mathbb{F}[x] \to \mathbb{F}$ for $i \in \mathbb{N}$ by

$$f = \sum_{i \in \mathbb{N}} c_i(f) x^i \quad \text{and} \quad e_i(f) = \Big(D^{\lfloor i/q \rfloor} f\Big)\Big(\omega_{i \bmod q}\Big) \quad \text{for } f \in \mathbb{F}[x],$$

where the residues modulo q are taken to be in $\{0, \ldots, q-1\}$. Then the Hermite interpolation problem that we consider in the remainder of the paper can be stated as follows: given $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$, compute the coefficients $c_0(f), \ldots, c_{\ell-1}(f)$ of the unique polynomial $f \in \mathbb{F}[x]_\ell$ that satisfies $e_i(f) = h_i$ for $i \in \{0, \ldots, \ell - 1\}$. The Hermite evaluation problem that we consider is the inverse problem, asking that we compute $e_0(f), \ldots, e_{\ell-1}(f)$ when given the coefficients $c_0(f), \ldots, c_{\ell-1}(f)$ of a polynomial $f \in \mathbb{F}[x]_\ell$. Existence and uniqueness for the interpolation problem follows from Lemma 2 with $n = \min(\ell, q)$ and multiplicities $\ell_i = \lceil (\ell - i)/q \rceil$ for $i \in \{0, \ldots, n-1\}$. We call ℓ the length of an instance of either problem, and observe that instances of length $\ell \le q$ 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 can be solved by existing standard interpolation and evaluation algorithms. The reductions employed by the algorithms are provided by the following lemma and the subsequent corollary.

Lemma 3. *Let $n \in \mathbb{N}$ be nonzero, $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and $f = f_1(x^q - x)^{2^{n-1}} + f_0$. Then*

$$\Big(D^i f\Big)(\omega) = \Big(D^i f_0\Big)(\omega) \quad \textit{and} \quad \Big(D^{2^{n-1}+i} f\Big)(\omega) = \Big(D^i\Big(f_1 + D^{2^{n-1}} f_0\Big)\Big)(\omega)$$

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

Proof. Let $n \in \mathbb{N}$ be nonzero, $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and $f = f_1(x^q - x)^{2^{n-1}} + f_0$. Then

$$f(x + \omega) = f_1(x + \omega)(x^q + \omega^q - x - \omega)^{2^{n-1}} + f_0(x + \omega) = f_1(x + \omega)\Big(x^{2^{n-1}q} + x^{2^{n-1}}\Big) + f_0(x + \omega)$$

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

$$\Big(D^i f\Big)(\omega) = \Big(D^i f_0\Big)(\omega) \quad \text{and} \quad \Big(D^{2^{n-1}+i} f\Big)(\omega) = \Big(D^i f_1\Big)(\omega) + \Big(D^{2^{n-1}+i} f_0\Big)(\omega)$$

for $\omega \in \mathbb{F}$ and $i \in \{0, \ldots, 2^{n-1} - 1\}$. Therefore, linearity of Hasse derivatives implies that the lemma will follow if we show that $D^{2^{n-1}+i} = D^i \circ D^{2^{n-1}}$ for $i \in \{0, \ldots, 2^{n-1} - 1\}$. To this end, we use Lucas' lemma (Lucas, 1878, p. 230; see also Fine, 1947), which states that

$$\binom{i}{j} \equiv \binom{\lfloor i/2^k \rfloor}{\lfloor j/2^k \rfloor}\binom{i \bmod 2^k}{j \bmod 2^k} \pmod 2 \quad \text{for } i, j, k \in \mathbb{N}. \tag{2}$$

By combining Lucas' lemma with Property 5 of Lemma 1, we find that

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

for $i \in \{0, \ldots, 2^{n-1} - 1\}$, as required. □

Corollary 4. *Let $n \in \mathbb{N}$ be nonzero, $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and $f = f_1(x^q - x)^{2^{n-1}} + f_0$. Then*

$$e_i(f) = e_i(f_0) \quad \textit{and} \quad e_{2^{n-1}q+i}(f) = e_i\Big(f_1 + D^{2^{n-1}} f_0\Big)$$

for $i \in \{0, \ldots, 2^{n-1}q - 1\}$.

Proof. Let $n \in \mathbb{N}$ be nonzero, $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ and $f = f_1(x^q - x)^{2^{n-1}} + f_0$. Then Lemma 3 implies that

$$e_i(f) = \Big(D^{\lfloor i/q \rfloor} f\Big)\Big(\omega_{i \bmod q}\Big) = \Big(D^{\lfloor i/q \rfloor} f_0\Big)\Big(\omega_{i \bmod q}\Big) = e_i(f_0)$$

and

$$e_{2^{n-1}q+i}(f) = \Big(D^{2^{n-1}+\lfloor i/q \rfloor} f\Big)\Big(\omega_{i \bmod q}\Big) = \Big(D^{\lfloor i/q \rfloor}\Big(f_1 + D^{2^{n-1}} f_0\Big)\Big)\Big(\omega_{i \bmod q}\Big) = e_i\Big(f_1 + D^{2^{n-1}} f_0\Big)$$

for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. □

For nonzero $n \in \mathbb{N}$ and $f \in \mathbb{F}[x]_{2^n q}$, there exist unique polynomials $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ such that $f = f_1(x^q - x)^{2^{n-1}} + f_0$. Corollary 4 then implies that the length $2^n q$ instance of the Hermite evaluation problem that corresponds to f can be replaced by the two length $2^{n-1}q$ instances that correspond to f_0 and $f_1 + D^{2^{n-1}} f_0$. Conversely, the corollary and Lemma 2 imply that the coefficients of the latter two polynomials are recovered by solving the two length $2^{n-1}q$ instances of the Hermite interpolation problem with inputs $e_0(f), \ldots, e_{2^{n-1}q-1}(f)$ and $e_{2^{n-1}q}(f), \ldots, e_{2^n q-1}(f)$. Combining the observation that

$$f = f_1(x^q - x)^{2^{n-1}} + f_0 = f_1 x^{2^{n-1}q} + f_1 x^{2^{n-1}} + f_0 \tag{3}$$

with the following lemma shows that only a linear number of additions are required to compute the coefficients of f_0 and $f_1 + D^{2^{n-1}} f_0$ when given those of f, and vice versa. It follows that instances of the interpolation and evaluation problems of length $2^n q$ for some nonzero $n \in \mathbb{N}$ may be reduced to two instances of length $2^{n-1}q$ at the cost of a linear number of additions.

Lemma 5. *Let $n \in \mathbb{N}$ be nonzero and $f \in \mathbb{F}[x]_{2^{n-1}q}$. Then*

$$D^{2^{n-1}} f = \sum_{i=0}^{q/2-1} x^{2^n i} \sum_{j=0}^{2^{n-1}-1} c_{2^{n-1}(2i+1)+j}(f) x^j. \tag{4}$$

Proof. Let $n \in \mathbb{N}$ be nonzero and $f \in \mathbb{F}[x]_{2^{n-1}q}$. Then Property 4 of Lemma 1 and Lucas' lemma, in the form of (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

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

and applying $D^{2^{n-1}}$ to each of its terms yields (4). □

By applying the reductions just described, we obtain Algorithms 1 and 2 for solving the Hermite interpolation and evaluation problems for instances of length $2^n q$ for some $n \in \mathbb{N}$. The algorithms recursively solve the half-length instances admitted by the reductions, and use the black-box algorithms Interpolate and Evaluate to solve the base case instances of length q. These algorithms must therefore satisfy the specifications of their parent algorithms for $n = 0$. Thus, they must perform standard interpolation or evaluation over the entire field, i.e., with evaluation points $\omega_0, \ldots, \omega_{q-1}$.

The algorithm HermiteInterpolate operates on a vector $(a_0, \ldots, a_{2^n q-1})$ of field elements that initially contains $e_0(f), \ldots, e_{2^n q-1}(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$, and overwrites its entries with the coefficients $c_0(f), \ldots, c_{2^n q-1}(f)$. If n is zero, then the vector is simply passed to the algorithm Interpolate. If n is positive, then the algorithm begins by making recursive calls on the left and right halves of the vector, replacing their entries by the coefficients of the polynomials f_0 and $f_1 + D^{2^{n-1}} f_0$, respectively. Lemma 5 implies that Lines 6 to 8 then add the coefficients of $D^{2^{n-1}} f_0$ to the corresponding coefficients of $f_1 + D^{2^{n-1}} f_0$ stored in the right half of the vector, so that it subsequently contains the coefficients of f_1. The expansion (3) then implies that Lines 9 and 10 set the entries of the vector equal to the coefficients of f, giving the correct output.

The algorithm HermiteEvaluate similarly operates on a vector $(a_0, \ldots, a_{2^n q-1})$ of field elements, which is passed directly to the algorithm Evaluate in the base case of $n = 0$. The recursive case of the algorithm simply reverses the steps performed in the recursive case of the interpolation algorithm by first computing the coefficients of f_0 and f_1, followed by those of $f_1 + D^{2^{n-1}} f_0$, then recursively evaluating f_0 and $f_1 + D^{2^{n-1}} f_0$.

The recursive cases of Algorithms 1 and 2 each perform $(3/4)2^n q$ additions on top of the recursive calls they make on the two halves of their input vector. It follows that if Interpolate or Evaluate performs a bounded number of operations for all inputs, say at most $A(q)$ additions and $M(q)$ multiplications, then its parent algorithm performs at most $2^n(A(q) + (3/4)qn)$ additions and $2^n M(q)$ multiplications overall. Interpolation and evaluation over the multiplicative group of the field can be performed with $O(\mathsf{M}(q-1))$ operations by the algorithms of Bostan and Schost (2005). Extending these algorithms to the entire field requires performing only a single extra addition in both cases, since $f \in \mathbb{F}[x]_q$ has residue $c_{q-1}(f) + \sum_{i=0}^{q-2} c_i(f) x^i \in \mathbb{F}[x]_{q-1}$ modulo $x^{q-1} - 1$, and $c_0(f) = e_i(f)$ for the index $i \in \{0, \ldots, q-1\}$ such that $\omega_i = 0$.

The subvectors of the input vector that appear in Algorithms 1 and 2 can be represented in practice by auxiliary variables, e.g., by a pointer to their first element. In doing so, the number of field elements stored in auxiliary space by either algorithm, i.e., in the space used by the algorithm in addition to the space required to store its inputs, is roughly equal to that required by a single instance of its base case algorithm.

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

Input: $n \in \mathbb{N}$, and $a_i = e_i(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$ and $i \in \{0, \ldots, 2^n q - 1\}$.
Output: $a_i = c_i(f)$ for $i \in \{0, \ldots, 2^n q - 1\}$.

```
If n = 0:
    Interpolate((a_0, ..., a_{q-1}))
Else:
    HermiteInterpolate(n − 1, (a_0, ..., a_{2^{n-1}q-1}))
    HermiteInterpolate(n − 1, (a_{2^{n-1}q}, ..., a_{2^n q-1}))
    For i = q/2, ..., q − 1:
        For j = 0, ..., 2^{n-1} − 1:
            a_{2^n i+j} ← a_{2^n i+j} + a_{2^n i+j−(q−1)2^{n-1}}
    For i = 2^{n-1}, ..., 2^{n-1}(q + 1) − 1:
        a_i ← a_i + a_{2^{n-1}(q−1)+i}
```

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

Input: $n \in \mathbb{N}$, and $a_i = c_i(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$ and $i \in \{0, \ldots, 2^n q - 1\}$.
Output: $a_i = e_i(f)$ for $i \in \{0, \ldots, 2^n q - 1\}$.

```
If n = 0:
    Evaluate((a_0, ..., a_{q-1}))
Else:
    For i = 2^{n-1}(q + 1) − 1, ..., 2^{n-1}:
        a_i ← a_i + a_{2^{n-1}(q−1)+i}
    For i = q/2, ..., q − 1:
        For j = 0, ..., 2^{n-1} − 1:
            a_{2^n i+j} ← a_{2^n i+j} + a_{2^n i+j−(q−1)2^{n-1}}
    HermiteEvaluate(n − 1, (a_0, ..., a_{2^{n-1}q-1}))
    HermiteEvaluate(n − 1, (a_{2^{n-1}q}, ..., a_{2^n q-1}))
```

Remark 6. If $\mathbb{F} = \mathbb{F}_q$ with q equal to a power of an odd prime p, then the Hermite interpolation and evaluation problems defined at the beginning of the section still make sense. Thus, it is natural to ask whether the techniques from this section can be applied to this case. If $f \in \mathbb{F}_q[x]_{p^n q}$ for some nonzero $n \in \mathbb{N}$, then there exist unique polynomials $f_0, \ldots, f_{p-1} \in \mathbb{F}_q[x]_{p^{n-1} q}$ such that

$$f = \sum_{i=0}^{p-1} (-1)^i f_i (x^q - x)^{p^{n-1} i}.$$

Generalising Lemma 3 then shows that solving the length $p^n q$ instances of the interpolation and evaluation problems that correspond to f can be reduced to solving the p length $p^{n-1} q$ instances of the problems that correspond to the polynomials $\sum_{j=0}^{i} D^{p^{n-1}(i-j)} f_j$ for $i \in \{0, \ldots, p-1\}$. For very small values of p, the later polynomials can be efficiently computed by naively generalising the approach used in this section. In doing so, the reduction may no longer be free of multiplications, since it may be necessary to perform scalar multiplications by elements of $\mathbb{F}_p$. For larger values of p, better methods of computing the polynomials are required to obtain efficient algorithms.

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 $c_0(f), \ldots, c_{2^n q-1}(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$, and $t \in \{1, \ldots, 2^n q\}$, compute $e_0(f), \ldots, e_{t-1}(f)$. The length ℓ Hermite evaluation problem is then captured by taking $n = \lceil \log_2 \lceil \ell/q \rceil \rceil$ and $t = \ell$.

The Hermite evaluation algorithm is described in Algorithm 4. The algorithm operates on a vector $(a_0, \ldots, a_{2^n q-1})$ of field elements that initially contains the coefficients of a polynomial $f \in \mathbb{F}[x]_{2^n q}$, and overwrites a_i with $e_i(f)$ for i less than the input value t. The remaining entries of the vector are either unchanged or set to intermediate values from the computation. If $t > 2^{n-1}q$, then the algorithm performs the same steps as Algorithm 2, with the exception that the recursive call used to evaluate $f_1 + D^{2^{n-1}} f_0$ only computes the $t - 2^{n-1}q$ values required for the output. If $t \leq 2^{n-1}q$, then the output depends on f_0 only, so only f_0 is computed (by the function PrepareLeft) and recursively evaluated. Once again, the recursion terminates with $n = 0$, which is handled by an algorithm TruncatedEvaluate that satisfies the specifications of Algorithm 3.

Algorithm 3 TruncatedEvaluate$(t, (a_0, \ldots, a_{q-1}))$

Input: $t \in \{1, \ldots, q\}$, and $a_i = c_i(f)$ for some $f \in \mathbb{F}[x]_q$ and $i \in \{0, \ldots, q-1\}$.
Output: $a_i = f(\omega_i)$ for $i \in \{0, \ldots, t-1\}$.

Algorithm 3 may be realised with a complexity of $O(\mathsf{M}(q))$ operations by using the method described in Section 3 to simply evaluate the polynomial over the entire field. This approach obviously wastes some effort when $t < q$, and, for example, may become less efficient than repeatedly apply Horner's rule when t is small. However, Algorithm 4 only ever makes at most one call to Algorithm 3 with $t < q$, so that it is not of critical importance in practice to optimise for this case unless q is large. In this case, if one has control over the enumeration of the field, then it may be beneficial to use truncated additive FFTs, which provide their best complexities when the degree of the field is smooth (Coxon, 2018b). Similarly, the truncated FFT of Larrieu (2017) can be used if the multiplicative group of the field has smooth order.

Proposition 7. *Algorithm 4 is correct if Algorithm 3 is correctly implemented.*

Proof. Suppose that Algorithm 3 has been correctly implemented. We use induction on the input parameter n to show that Algorithm 4 is correct under this assumption. For inputs with $n = 0$, the algorithm trivially produces the correct output since Algorithm 3 is simply applied in this case. Therefore, suppose that the algorithm is called with a nonzero input n, and that the algorithm produces the correct output for all inputs with smaller values of n. Let $f \in \mathbb{F}[x]_{2^n q}$ be the polynomial that corresponds to the input, and $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ be the unique polynomials such that $f = f_1(x^q - x)^{2^{n-1}} + f_0$.

Suppose that $t > 2^{n-1}q$. Then, as (3) holds, Lines 4 and 5 set $a_i = c_i(f_0)$ and $a_{2^{n-1}q+i} = c_i(f_1)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Consequently, Lemma 5 implies that Lines 6 to 8 set $a_{2^{n-1}q+i} = c_i(f_1 + D^{2^{n-1}} f_0)$ 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 Corollary 4 imply that the recursive call of Line 9 sets $a_i = e_i(f_0) = e_i(f)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Similarly, the recursive

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

Input: $n \in \mathbb{N}$, $t \in \{1, \ldots, 2^n q\}$, and $a_i = c_i(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$ and $i \in \{0, \ldots, 2^n q - 1\}$.
Output: $a_i = e_i(f)$ for $i \in \{0, \ldots, t-1\}$.

1: If $n = 0$:
2: TruncatedEvaluate$(t, (a_0, \ldots, a_{q-1}))$ */* Algorithm 3 */*
3: Else if $t > 2^{n-1}q$:
4: For $i = 2^{n-1}(q+1) - 1, \ldots, 2^{n-1}$:
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: TruncatedHermiteEvaluate$(n-1, 2^{n-1}q, (a_0, \ldots, a_{2^{n-1}q-1}))$
10: TruncatedHermiteEvaluate$(n-1, t - 2^{n-1}q, (a_{2^{n-1}q}, \ldots, a_{2^n q-1}))$
11: Else:
12: PrepareLeft$(n, 0, (a_0, \ldots, a_{2^n q-1}))$
13: TruncatedHermiteEvaluate$(n-1, t, (a_0, \ldots, a_{2^{n-1}q-1}))$

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

call of Line 10 sets $a_{2^{n-1}q+i} = e_i(f_1 + D^{2^{n-1}} f_0) = e_{2^{n-1}q+i}(f)$ for $i \in \{0, \ldots, t - 2^{n-1}q - 1\}$. The algorithm stops at this point, and thus produces the correct output.

Suppose now that $t \leq 2^{n-1}q$. Then, as (3) holds, Line 12 sets $a_i = c_i(f_0)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Consequently, the induction hypothesis and Corollary 4 imply that the recursive call of Line 13 sets $a_i = e_i(f_0) = e_i(f)$ for $i \in \{0, \ldots, t-1\}$. The algorithm stops at this point, and thus produces the correct output. □

For nonzero $i, j \in \mathbb{N}$, define $i \bmod^* j = i - (\lceil i/j \rceil - 1)j$, the residue of i modulo j that lies in $\{1, \ldots, j\}$. The following proposition bounds the additive and multiplicative complexities of Algorithm 4 in terms of those of Algorithm 3.

Proposition 8. *Suppose there exist functions $A, M : \{1, \ldots, q\} \to \mathbb{N}$ such that Algorithm 3 performs at most $A(t)$ additions and at most $M(t)$ multiplications (in $\mathbb{F}$) when called with input parameter t. Then Algorithm 4 performs at most*

$$A(q)(\lceil t/q \rceil - 1) + A(t \bmod^* q) + \left(\frac{3}{4}\lceil \log_2 \lceil t/q \rceil \rceil - \frac{1}{4}\right)(\lceil t/q \rceil - 1)q + (2^n - 1)q \tag{5}$$

additions and at most $M(q)(\lceil t/q \rceil - 1) + M(t \bmod^* q)$ multiplications.

Proof. Suppose there exists functions $A, M : \{1, \ldots, q\} \to \mathbb{N}$ that satisfy the conditions of the proposition. We use induction on the input parameter n to show that the stated bounds on the number of additions and multiplications performed by Algorithm 4 then hold. If $n = 0$, then the bounds hold trivially since Algorithm 4 simply passes its input vector to Algorithm 3, and the

input parameter t is at most q. Therefore, suppose that the algorithm is called with a nonzero input n, and that the bounds stated in the proposition hold for all inputs with smaller values of n. Assume, to begin with, that $t > 2^{n-1}q$. Then Lines 4 to 8 of the algorithm perform $(3/4)2^n q$ additions and no multiplications. Line 9 then performs at most $2^{n-1}(A(q)+(3/4)q(n-1))$ additions and at most $2^{n-1}M(q)$ multiplications, since the algorithm reduces to Algorithm 2 for inputs with $t = 2^n q$. As $2^{n-1}q < t \leq 2^n q$, we have

$$\left(\frac{3}{4}\left\lceil \log_2\lceil t/q\rceil - 2^{n-1}\right\rceil - \frac{1}{4}\right)\left(\lceil t/q\rceil - 2^{n-1} - 1\right) \leq \left(\frac{3}{4}n - \frac{1}{4}\right)(\lceil t/q\rceil - 1) - \frac{3}{4}2^{n-1} - \left(\frac{3}{4}n - 1\right)2^{n-1}$$
$$= \left(\frac{3}{4}\lceil \log_2\lceil t/q\rceil\rceil - \frac{1}{4}\right)(\lceil t/q\rceil - 1) - \left(\frac{3}{4}n - \frac{1}{4}\right)2^{n-1}.$$

Consequently, the induction hypothesis implies that Line 10 performs at most

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

additions, and most $M(q)(\lceil t/q\rceil - 2^{n-1} - 1) + M(t \bmod^* q)$ multiplications. Summing these two sets of bounds shows that the two bounds stated in the proposition are satisfied if $t > 2^{n-1}q$.

If $t \leq 2^{n-1}q$, then Line 12 of Algorithm 4 performs $2^{n-1}q$ additions and no multiplications. Combining these contributions with the bounds on the number of additions and multiplications performed by Line 13 provided by the induction hypothesis shows that the bounds stated in the proposition are satisfied if $t \leq 2^{n-1}q$. □

When $n = 0$ or $t > 2^{n-1}q$ for some nonzero n, as may be assumed when solving the Hermite evaluation problem, the third and fourth terms of the bound (5) are in $O(t \log\lceil t/q\rceil)$. By using the method of evaluating polynomials described in Section 3 to realise Algorithm 3, it follows that the length ℓ Hermite evaluation problem can be solved with $O(\lceil \ell/q\rceil \mathsf{M}(q) + \ell \log\lceil \ell/q\rceil)$ operations by Algorithm 4. In this setting, it is possible to reduce the number of additions performed by Algorithm 4 when ℓ is not of the form $2^n q$ by adapting the algorithm to take into account the $2^n q - \ell$ zeros that initially occupy the rightmost entries of the input vector.

5. Interpolation algorithm

To solve the Hermite interpolation problem for arbitrary lengths, we follow the approach introduced by van der Hoeven (2004) for his inverse truncated FFT (see also Harvey, 2009; Larrieu, 2017) 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 $t \in \{1, \ldots, 2^n q\}$ and $h_0, \ldots, h_{t-1}, f_t, \ldots, f_{2^n q-1} \in \mathbb{F}$, compute $c_0(f), \ldots, c_{t-1}(f)$ for the unique polynomial $f \in \mathbb{F}[x]_{2^n q}$ such that $e_i(f) = h_i$ for $i \in \{0, \ldots, t-1\}$ and $c_i(f) = f_i$ for $i \in \{t, \ldots, 2^n q - 1\}$. Existence and uniqueness of the polynomial f are readily shown to follow from Lemma 2. The length ℓ Hermite interpolation problem is then captured as an instance of the new problem by taking $n = \lceil \log_2\lceil \ell/q\rceil\rceil$, $t = \ell$ and $f_t = \cdots = f_{2^n q-1} = 0$.

The Hermite interpolation algorithm is described in Algorithm 6. If $t > 2^{n-1}q$, 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 5 and the expansion (3) to compute the

desired coefficients of f. However, the recursive call that is used to recover $f_1 + D^{2^{n-1}} f_0$ cannot be made without first computing $c_i(f_1 + D^{2^{n-1}} f_0)$ for $i \geq t - 2^{n-1}q$. 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 $i \geq t$. If $t \leq 2^{n-1}q$, then the function PrepareLeft from Algorithm 4 is used to recover $c_i(f_0)$ for $i \geq t$, before the remaining coefficients of the polynomial are recursively computed. The function PrepareLeft, which is its own inverse for fixed t, is then used to compute the lower order coefficients of the output. The base case of the recursion is handled by an algorithm TruncatedInterpolate that satisfies the specifications of Algorithm 5.

Algorithm 5 TruncatedInterpolate$(t, (a_0, \ldots, a_{q-1}))$

Input: $t \in \{1, \ldots, q\}$, $a_i = f(\omega_i)$ for some $f \in \mathbb{F}[x]_q$ and $i \in \{0, \ldots, t-1\}$, and $a_i = c_i(f)$ for $i \in \{t, \ldots, q-1\}$.
Output: $a_i = c_i(f)$ for $i \in \{0, \ldots, q-1\}$.

Algorithm 5 must compute the coefficients of the polynomial $\sum_{i=0}^{t-1} c_i(f)x^i$, which may be recovered by first evaluating $g = \sum_{i=t}^{q-1} c_i(f)x^i$ at the points $\omega_0, \ldots, \omega_{t-1}$, then interpolating the values $f(\omega_i) + g(\omega_i)$ for $i \in \{0, \ldots, t-1\}$. The evaluation step can be performed with $O(\mathsf{M}(q))$ operations by simply evaluating g over the entire field, while the interpolation step can be performed with $O(\mathsf{M}(t)\log t)$ operations by fast Chinese remainder algorithms, or $O(\mathsf{M}(t))$ operations by the algorithm of Bostan and Schost (2005) when $\omega_0, \ldots, \omega_{t-1}$ is a geometric progression. It follows that Algorithm 5 may be realised with a complexity of $O(\mathsf{M}(q) + \mathsf{M}(t)\log t)$ operations in $\mathbb{F}$. In the special case $t = q$, which holds for all but at most one call to the algorithm by Algorithm 6, the algorithm only requires $O(\mathsf{M}(q))$ operations since g is zero. Moreover, given control over the enumeration of the field, we can obtain this complexity bound for all values of t by taking $\omega_0, \ldots, \omega_{q-2}$ to be a geometric progression. In this case, Algorithm 5 may also be realised by using inverse truncated FFTs (Larrieu, 2017; Coxon, 2018b), which may be beneficial when q is large and the degree of the field, or the order of its multiplicative group, is smooth.

Proposition 9. *Algorithm 6 is correct if Algorithm 5 is correctly implemented.*

Proof. Suppose that Algorithm 5 has been correctly implemented. We use induction on the input parameter n to show that Algorithm 6 is correct under this assumption. For inputs with $n = 0$, the algorithm trivially produces the correct output since Algorithm 5 is simply applied in this case. Therefore, suppose that the algorithm is called with a nonzero input n, and that the algorithm produces the correct output for all inputs with smaller values of n. Let $f \in \mathbb{F}[x]_{2^n q}$ be the polynomial that corresponds to the input, and $f_0, f_1 \in \mathbb{F}[x]_{2^{n-1}q}$ be the unique polynomials such that $f = f_1(x^q - x)^{2^{n-1}} + f_0$.

Suppose that $t > 2^{n-1}q$. Then Corollary 4 implies that at the beginning of the algorithm, $a_i = e_i(f) = e_i(f_0)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. Consequently, the induction hypothesis and Lemma 2 imply that the recursive call of Line 4 sets $a_i = c_i(f_0)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. As (3) holds, it follows that when the function PrepareRight is called in Line 5, Lines 17 and 18 of the function set $a_{2^{n-1}q+i} = c_i(f_1)$ for $i \in \{t - 2^{n-1}q, \ldots, 2^{n-1}q - 1\}$. Then Lemma 5 implies that Lines 19 to 25 of the function set $a_{2^{n-1}q+i} = c_i(f_1 + D^{2^{n-1}} f_0)$ for $i \in \{t - 2^{n-1}q, \ldots, 2^{n-1}q - 1\}$. The entries $a_{2^{n-1}q}, \ldots, a_{t-1}$ are so far unchanged by the algorithm. Thus, Corollary 4 implies that $a_{2^{n-1}q+i} = e_{2^{n-1}q+i}(f) = e_i(f_1 + D^{2^{n-1}} f_0)$ for $i \in \{0, \ldots, t - 2^{n-1}q - 1\}$. The induction hypothesis and Lemma 2

Algorithm 6 TruncatedHermiteInterpolate$(n, t, (a_0, \ldots, a_{2^n q-1}))$

Input: $n \in \mathbb{N}$, $t \in \{1, \ldots, 2^n q\}$, $a_i = e_i(f)$ for some $f \in \mathbb{F}[x]_{2^n q}$ and $i \in \{0, \ldots, t-1\}$, and $a_i = c_i(f)$ for $i \in \{t, \ldots, 2^n q - 1\}$.

Output: $a_i = c_i(f)$ for $i \in \{0, \ldots, 2^n q - 1\}$.

1: If $n = 0$:
2: TruncatedInterpolate$(t, (a_0, \ldots, a_{q-1}))$ */* Algorithm 5 */*
3: Else if $t > 2^{n-1}q$:
4: TruncatedHermiteInterpolate$(n-1, 2^{n-1}q, (a_0, \ldots, a_{2^{n-1}q-1}))$
5: PrepareRight$(n, t, (a_0, \ldots, a_{2^n q-1}))$
6: TruncatedHermiteInterpolate$(n-1, t-2^{n-1}q, (a_{2^{n-1}q}, \ldots, a_{2^n q-1}))$
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$(n, t, (a_0, \ldots, a_{2^n q-1}))$ */* From Algorithm 4 */*
14: TruncatedHermiteInterpolate$(n-1, t, (a_0, \ldots, a_{2^{n-1}q-1}))$
15: PrepareLeft$(n, 0, (a_0, \ldots, a_{2^n q-1}))$

16: **Function** PrepareRight$(n, t, (a_0, \ldots, a_{2^n q-1}))$:
17: For $i = 2^{n-1}(q+1)-1, 2^{n-1}(q+1)-2, \ldots, t$:
18: $a_i \leftarrow a_i + a_{2^{n-1}(q-1)+i}$
19: $s \leftarrow \lfloor t/2^n \rfloor$, $r \leftarrow \min(t \bmod 2^n, 2^{n-1})$
20: For $j = 0, \ldots, r-1$:
21: For $i = s+1, \ldots, q-1$:
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 $i = s, \ldots, q-1$:
25: $a_{2^n i+j} \leftarrow a_{2^n i+j} + a_{2^n i+j-(q-1)2^{n-1}}$

therefore imply that the recursive call of Line 6 sets $a_{2^{n-1}q+i} = c_i(f_1 + D^{2^{n-1}} f_0)$ for $i \in \{0, \ldots, t - 2^{n-1}q - 1\}$. 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 5 implies that Lines 7 to 9 set $a_{2^{n-1}q+i} = c_i(f_1)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$, then (3) implies that Lines 10 to 11 set $a_i = c_i(f)$ for $i \in \{0, \ldots, 2^n q - 1\}$. The algorithm stops at this point, and thus produces the correct output.

Suppose now that $t \leq 2^{n-1}q$. Then, as (3) holds, the call to PrepareLeft in Line 13 sets $a_i = c_i(f_0)$ for $i \in \{t, \ldots, 2^{n-1}q - 1\}$. This call to PrepareLeft does not modify $a_0, \ldots, a_{t-1}$. Thus, Corollary 4 implies that $a_i = e_i(f) = e_i(f_0)$ for $i \in \{0, \ldots, t-1\}$ when the recursive call of Line 14 is made. The induction hypothesis and Lemma 2 therefore imply that Line 14 sets $a_i = c_i(f_0)$ for $i \in \{0, \ldots, t-1\}$. 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 $a_i = c_i(f)$ for $i \in \{2^{n-1}q, \ldots, 2^n q - 1\}$. Thus, (3) implies that the call to PrepareLeft in

Line 15 sets $a_i = c_i(f)$ for $i \in \{0, \ldots, 2^{n-1}q - 1\}$. The algorithm stops at this point, and thus produces the correct output. □

Proposition 10. *Suppose there exist functions $A, M : \{1, \ldots, q\} \to \mathbb{N}$ such that Algorithm 5 performs at most $A(t)$ additions and at most $M(t)$ multiplications (in $\mathbb{F}$) when called with input parameter t. Then Algorithm 6 performs at most*

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

additions and at most $M(q)(\lceil t/q \rceil - 1) + M(t \bmod^* q)$ multiplications.

Proof. We assume the existence of functions $A, M : \{1, \ldots, q\} \to \mathbb{N}$ that satisfy the conditions of the proposition, and use induction on the parameter n to prove the state bounds. The bounds hold trivially if $n = 0$, since Algorithm 6 simply passes its input vector to Algorithm 5 in this case. Therefore, suppose that the algorithm is called with a nonzero input n, and that the bounds stated in the proposition hold for all inputs with smaller values of n. Assume, to begin with, that $t > 2^{n-1}q$. Then the call to PrepareRight in Line 5 of Algorithm 6 performs

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

additions, where $s = \lfloor t/2^n \rfloor$ and $r = \min(t \bmod 2^n, 2^{n-1})$. In particular, no additions are performed if $t = 2^n q$. It follows that the algorithm reduces to Algorithm 1 for inputs with $t = 2^n q$. Thus, Line 4 performs at most $2^{n-1}(A(q) + (3/4)q(n - 1))$ additions and at most $2^{n-1}M(q)$ multiplications. As $2^{n-1}q < t \leq 2^n q$, we have

$$\left(\frac{7}{4}\left\lceil \log_2 \lceil t/q \rceil - 2^{n-1} - 1 \right\rceil - n - \frac{3}{4}\right)\left(\lceil t/q \rceil - 2^{n-1} - 1\right) \leq \left(\frac{7}{4}\lceil \log_2 \lceil t/q \rceil \rceil - n - \frac{3}{4}\right)(\lceil t/q \rceil - 1) - \frac{3}{4}2^{n-1}(n - 1).$$

Consequently, the induction hypothesis implies that Line 6 performs at most

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

additions, and at most $M(q)(\lceil t/q \rceil - 2^{n-1} - 1) + M(t \bmod^* q)$ multiplications. Finally, Lines 7 to 11 perform $(3/4)2^n q$ additions and no multiplications. Summing these two sets of bounds shows that the two bounds stated in the proposition are satisfied if $t > 2^{n-1}q$.

If $t \leq 2^{n-1}q$, then Line 13 of Algorithm 6 performs at most $(2^{n-1}q - t) + (2^n - 2^{n-1})$ additions, while Line 15 performs $2^{n-1}q$ additions. Thus, Line 13 and 15 perform at most $2^{n-1}(2q + 1) - t < 2^{n-1}(2q + 1) - (\lceil t/q \rceil - 1)q$ additions and no multiplications. Combining these bounds with those provided by the induction hypothesis on the number of additions and multiplications performed by Line 14 then shows that the bounds stated in the proposition are satisfied if $t \leq 2^{n-1}q$. □

Using the methods described earlier in the section, Algorithm 5 may be realised so that $O(\lceil t/q \rceil \mathsf{M}(q) + \mathsf{M}(t \bmod^* q) \log(t \bmod^* q))$ operations are performed over all calls to the algorithm made by Algorithm 6. Consequently, Proposition 10 implies that the length ℓ Hermite interpolation problem may be solved by Algorithm 6 with $O((\lceil \ell/q \rceil + \log q)\mathsf{M}(q) + \ell \log \lceil \ell/q \rceil)$ operations. Given control over the enumeration of the field, we showed that Algorithm 5 may be realised with a complexity of $O(\mathsf{M}(q))$ operations. For this case, the proposition implies that the length ℓ Hermite interpolation problem may be solved with $O(\lceil \ell/q \rceil \mathsf{M}(q) + \ell \log \lceil \ell/q \rceil)$ operations by Algorithm 6. The number of additions performed by the algorithm in this setting may be reduced by taking into account the $2^n q - \ell$ zeros that initially occupy the rightmost entries of the input vector. 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.

Acknowledgements

This work was supported by Nokia in the framework of the common laboratory between Nokia Bell Labs and INRIA. The author is grateful for the comments and suggestions of the anonymous reviewer, which greatly improved the presentation and results of the paper. The author would also like to thank David Pearce, Sian Stafford, Andrew Calcino and Paulina Tapia for their gracious accommodation during the preparation of this work.

Aho, A. V., Steiglitz, K., Ullman, J. D., 1975. Evaluating polynomials at fixed sets of points. SIAM J. Comput. 4 (4), 533–539.
URL http://dx.doi.org/10.1137/0204045

Augot, D., Levy-dit-Vehel, F., Shikfa, A., 2014. A storage-efficient and robust private information retrieval scheme allowing few servers. In: Cryptology and network security. Vol. 8813 of Lecture Notes in Comput. Sci. Springer, Cham, pp. 222–239.
URL http://dx.doi.org/10.1007/978-3-319-12280-9_15

Bernstein, D. J., 2004. Scaled remainder trees, Available from https://cr.yp.to/arith/scaledmod-20040820.pdf.
URL https://cr.yp.to/arith/scaledmod-20040820.pdf

Bini, D., Pan, V. Y., 1994. Polynomial and matrix computations, vol. 1: Fundamental algorithms. Progress in Theoretical Computer Science. Birkhäuser Boston, Inc., Boston, MA.
URL http://dx.doi.org/10.1007/978-1-4612-0265-3

Borodin, A., Moenck, R., 1974. Fast modular transforms. J. Comput. System Sci. 8, 366–386.

Bostan, A., Lecerf, G., Salvy, B., Schost, E., Wiebelt, B., 2004. Complexity issues in bivariate polynomial factorization. In: ISSAC 2004. ACM, New York, pp. 42–49.
URL https://doi.org/10.1145/1005285.1005294

Bostan, A., Lecerf, G., Schost, E., 2003. Tellegen's principle into practice. In: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation. ISSAC '03. ACM, New York, NY, USA, pp. 37–44.
URL http://doi.acm.org/10.1145/860854.860870

Bostan, A., Schost, É., 2005. Polynomial evaluation and interpolation on special sets of points. J. Complexity 21 (4), 420–446.
URL http://dx.doi.org/10.1016/j.jco.2004.09.009

Cantor, D. G., Kaltofen, E., 1991. On fast multiplication of polynomials over arbitrary algebras. Acta Inform. 28 (7), 693–701.
URL http://dx.doi.org/10.1007/BF01178683

Chin, F. Y., 1976. A generalized asymptotic upper bound on fast polynomial evaluation and interpolation. SIAM J. Comput. 5 (4), 682–690.
URL http://dx.doi.org/10.1137/0205047

Coxon, N., 2018a. Fast systematic encoding of multiplicity codes. J. Symbolic Comput. 94, 234–254.
URL https://doi.org/10.1016/j.jsc.2018.08.005

Coxon, N., July 2018b. Fast transforms over finite fields of characteristic two, arXiv:1807.07785 [cs.SC].
URL https://arxiv.org/abs/1807.07785

Fiduccia, C. M., 1972. Polynomial evaluation via the division algorithm: the fast Fourier transform revisited. In: Proceedings of the Fourth Annual ACM Symposium on Theory of Computing. STOC '72. ACM, New York, NY, USA, pp. 88–93.
URL http://doi.acm.org/10.1145/800152.804900

Fine, N. J., 1947. Binomial coefficients modulo a prime. Amer. Math. Monthly 54, 589–592.
URL http://dx.doi.org/10.2307/2304500

Gao, S., Mateer, T., 2010. Additive fast Fourier transforms over finite fields. IEEE Trans. Inform. Theory 56 (12), 6265–6272.
URL http://dx.doi.org/10.1109/TIT.2010.2079016

Harvey, D., 2009. A cache-friendly truncated FFT. Theoret. Comput. Sci. 410 (27-29), 2649–2658.
URL https://doi.org/10.1016/j.tcs.2009.03.014

Harvey, D., Roche, D. S., 2010. An in-place truncated Fourier transform and applications to polynomial multiplication. In: ISSAC 2010—Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 325–329.
URL http://dx.doi.org/10.1145/1837934.1837996

Harvey, D., van der Hoeven, J., December 2017. Faster integer and polynomial multiplication using cyclotomic coefficient rings, arXiv:1712.03693 [cs.SC].
URL http://arxiv.org/abs/1712.03693

Harvey, D., van der Hoeven, J., Lecerf, G., 2016. Fast polynomial multiplication over $\mathbb{F}_{2^{60}}$. In: Proc. ISSAC '16. ACM, New York, NY, USA, pp. 255–262.

Harvey, D., van der Hoeven, J., Lecerf, G., 2017. Faster polynomial multiplication over finite fields. J. ACM 63 (6), 52:1–52:23.

Kopparty, S., 2014. Some remarks on multiplicity codes. In: Discrete geometry and algebraic combinatorics. Vol. 625 of Contemp. Math. Amer. Math. Soc., Providence, RI, pp. 155–176.
URL https://doi.org/10.1090/conm/625/12497

Larrieu, R., 2017. The truncated Fourier transform for mixed radices. In: Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation. ISSAC'17. ACM, New York, NY, USA, pp. 261–268.
URL http://doi.acm.org/10.1145/3087604.3087636

Lucas, E., 1878. Théorie des Fonctions Numériques Simplement Périodiques. [Continued]. Amer. J. Math. 1 (3), 197–240.
URL http://dx.doi.org/10.2307/2369311

Markel, J., Dec 1971. FFT pruning. IEEE Transactions on Audio and Electroacoustics 19 (4), 305–311.

Moenck, R., Borodin, A., 1972. Fast modular transforms via division. In: Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972). SWAT '72. IEEE Computer Society, Washington, DC, USA, pp. 90–96.
URL https://doi.org/10.1109/SWAT.1972.5

Olshevsky, V., Shokrollahi, A., 2000. Matrix-vector product for confluent Cauchy-like matrices with application to confluent rational interpolation. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing. ACM, New York, pp. 573–581.
URL http://dx.doi.org/10.1145/335305.335380

Pan, V. Y., 2001. Structured matrices and polynomials: unified superfast algorithms. Birkhäuser Boston, Inc., Boston, MA; Springer-Verlag, New York.
URL http://dx.doi.org/10.1007/978-1-4612-0129-8

Schönhage, A., 1976/77. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. Acta Informat. 7 (4), 395–398.

Schönhage, A., Strassen, V., 1971. Schnelle Multiplikation grosser Zahlen. Computing (Arch. Elektron. Rechnen) 7, 281–292.

Sorensen, H. V., Burrus, C. S., 1993. Efficient computation of the DFT with only a subset of input or output points. IEEE Transactions on Signal Processing 41 (3), 1184–1200.

van der Hoeven, J., 2004. The truncated Fourier transform and applications. In: ISSAC 2004. ACM, New York, pp. 290–296.
URL http://dx.doi.org/10.1145/1005285.1005327

van der Hoeven, J., 2005. Notes on the Truncated Fourier Transform. Tech. Rep. 2005-5, Université Paris-Sud, Orsay, France.

van der Hoeven, J., 2016. Faster Chinese remaindering. Tech. rep., HAL, http://hal.archives-ouvertes.fr/hal-01403810.

Vari, T. M., 1974. Some complexity results for a class of Toeplitz matrices. Tech. rep., Dept. of Computer Sci. and Math., York Univ., Toronto.

von zur Gathen, J., 1990. Functional decomposition of polynomials: the tame case. J. Symbolic Comput. 9 (3), 281–299.

URL http://dx.doi.org/10.1016/S0747-7171(08)80014-4
von zur Gathen, J., Gerhard, J., 1997. Fast algorithms for Taylor shifts and certain difference equations. In: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 40–47.
URL http://dx.doi.org/10.1145/258726.258745
von zur Gathen, J., Gerhard, J., 2013. Modern computer algebra, 3rd Edition. Cambridge University Press, Cambridge.
URL http://dx.doi.org/10.1017/CBO9781139856065
Woodruff, D., Yekhanin, S., 2007. A geometric approach to information-theoretic private information retrieval. SIAM J. Comput. 37 (4), 1046–1056.
URL https://doi.org/10.1137/06065773X
Wu, L., September 2015. Revisiting the multiplicity codes: A new class of high-rate locally correctable codes. In: 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton). pp. 509–513.