Fast polynomial factorization and modular composition[*]

Kiran S. Kedlaya[†]
MIT

Christopher Umans[‡]
Caltech

June 13, 2011

Abstract

We obtain randomized algorithms for factoring degree n univariate polynomials over $\mathbb{F}_q$ requiring $O(n^{1.5+o(1)} \log^{1+o(1)} q + n^{1+o(1)} \log^{2+o(1)} q)$ bit operations. When $\log q < n$, this is asymptotically faster than the best previous algorithms (von zur Gathen & Shoup (1992) and Kaltofen & Shoup (1998)); for $\log q \geq n$, it matches the asymptotic running time of the best known algorithms.

The improvements come from new algorithms for modular composition of degree n univariate polynomials, which is the asymptotic bottleneck in fast algorithms for factoring polynomials over finite fields. The best previous algorithms for modular composition use $O(n^{(\omega+1)/2})$ field operations, where ω is the exponent of matrix multiplication (Brent & Kung (1978)), with a slight improvement in the exponent achieved by employing fast rectangular matrix multiplication (Huang & Pan (1997)).

We show that modular composition and multipoint evaluation of multivariate polynomials are essentially equivalent, in the sense that an algorithm for one achieving exponent α implies an algorithm for the other with exponent $\alpha + o(1)$, and vice versa. We then give two new algorithms that solve the problem near-optimally: an algebraic algorithm for fields of characteristic at most $n^{o(1)}$, and a nonalgebraic algorithm that works in arbitrary characteristic. The latter algorithm works by lifting to characteristic 0, applying a small number of rounds of *multimodular reduction*, and finishing with a small number of multidimensional FFTs. The final evaluations are reconstructed using the Chinese Remainder Theorem. As a bonus, this algorithm produces a very efficient data structure supporting polynomial evaluation queries, which is of independent interest.

Our algorithms use techniques that are commonly employed in practice, in contrast to all previous subquadratic algorithms for these problems, which relied on fast matrix multiplication.

[*]The material in this paper appeared in conferences as [Uma08] and [KU08].

[†]Supported by NSF DMS-0545904 (CAREER) and a Sloan Research Fellowship.

[‡]Supported by NSF CCF-0346991 (CAREER), CCF-0830787, BSF 2004329, and a Sloan Research Fellowship.

1 Introduction

Polynomial factorization is one of the central problems in computer algebra. Milestones in the development of polynomial-time algorithms for factoring in $\mathbb{F}_q[X]$ are the algorithms of Berlekamp [Ber70], Cantor & Zassenhaus [CZ81], von zur Gathen & Shoup [vzGS92] and Kaltofen & Shoup [KS98]. See the surveys [vzGP01, Kal03, vzG06]. Presently, there are practical algorithms that factor degree n polynomials over $\mathbb{F}_q$ using a quadratic number of operations (ignoring for a moment the dependence on q), and subquadratic algorithms that rely on fast matrix multiplication [KS98]. Efficient algorithms for factoring polynomials over other domains (e.g., $\mathbb{Q}$, $\mathbb{Z}$, algebraic number fields) and for factoring multivariate polynomials in turn depend on factoring in $\mathbb{F}_q[X]$.

The bottleneck in most modern factoring algorithms (including the asymptotically fastest ones) turns out to be the computation of the "Frobenius power" polynomials, X^{q^i}, modulo the degree n polynomial h to be factored, for various i between 1 and n. When $i = n$, a repeated-squaring approach requires $n \log q$ modular multiplications of degree n polynomials. A clever improvement based on the so-called "polynomial representation of the Frobenius map" (an idea attributed to Kaltofen) was exploited in this context by von zur Gathen & Shoup [vzGS92]: first compute $X^q \bmod h(X)$ by repeated squaring, then *compose* that polynomial with itself modulo $h(X)$ to get

$$(X^q)^q \bmod h(X) = X^{q^2} \bmod h(X).$$

Repeating the composition $\log n$ times produces $X^{q^n} \bmod h(X)$ with only $\log q$ modular multiplications and $\log n$ modular compositions overall. There are subquadratic algorithms for modular composition, and so this approach is asymptotically superior to the straightforward repeated-squaring algorithm. The same idea can also be applied to other problems that arise in polynomial factorization, like computing the norm and trace maps, $X^{q^{n-1}+q^{n-2}+\cdots+q+1}$ and $X^{q^{n-1}} + X^{q^{n-2}} + \cdots + X^q + X$, with similar speedups.

Thus the modular composition problem emerges as a crucial component of the fastest factoring algorithms (as well as other problems, such as irreducibility testing and constructing irreducible polynomials [Sho94], and manipulating normal bases of finite fields [KS98]). Indeed, if we could compute $f(g(X)) \bmod h(X)$ for degree n polynomials $f, g, h \in \mathbb{F}_q[X]$ in n^α operations, then there are algorithms for factoring degree n polynomial over $\mathbb{F}_q$ using

$$O(n^{\alpha+1/2+o(1)} + n^{1+o(1)} \log q)$$

operations. For comparison, the currently fastest algorithms take either $O(n^2 + n \log q) \cdot \operatorname{poly}\log(n, \log q)$ [vzGS92] or $O(n^{1.815} \log q) \cdot \operatorname{poly}\log(n, \log q)$ [KS98] operations (also, see the more precise accounting and detailed comparisons in Figure 1 of [KS98]).

1.1 Modular composition of polynomials

The problem of modular composition is, given three degree n univariate polynomials $f(x), g(x), h(x)$ over a ring with h having invertible leading coefficient, to compute $f(g(x)) \pmod{h(x)}$. In contrast to other basic modular operations on polynomials (e.g., modular multiplication), it is *not* possible to obtain an asymptotically fast algorithm for modular composition with fast algorithms for each step in the natural two step procedure (i.e., first compute $f(g(x))$, then reduce modulo $h(x)$). This is because $f(g(x))$ has n^2 terms, while we hope for a modular composition algorithm that uses only about $O(n)$ operations. Not surprisingly, it is by considering the overall operation (and beating n^2) that asymptotic gains are made in algorithms that employ modular composition.

Perhaps because nontrivial algorithms for modular composition must handle the modulus in an integrated way (rather than computing a remainder after an easier, nonmodular computation) there have been few algorithmic inroads on this seemingly basic problem. Brent & Kung [BK78] gave the first nontrivial algorithm in 1978, achieving an operation count of $O(n^{(\omega+1)/2})$, where ω is the exponent of matrix multiplication (the best upper bound is currently $\omega < 2.376$ [CW90]). Huang & Pan [HP98] achieved a small improvement, by noting that the bound is actually $O(n^{\omega_2/2})$ where ω_2 is the exponent of $n \times n$ by $n \times n^2$ matrix multiplication, and giving an upper bound on ω_2 that is slightly better than $2.376 + 1$. Even with optimal matrix multiplication, these algorithms cannot beat $O(n^{1.5})$, and it is currently not feasible in practice to achieve their theoretical guarantees, because those rely on the asymptotically fastest algorithms for matrix multiplication, which are currently impractical. Finding new algorithms for MODULAR COMPOSITION with running times closer to $O(n)$ was mentioned several times as an important and longstanding open problem (cf. [Sho94, KS98], [BCS97, Problem 2.4], [vzGG99, Research Problem 12.19]).

We note that the special case of modular composition in which the modulus $h(X)$ is X^d has an algorithm attributed to Brent & Kung that uses $O(n^{1.5}) \cdot \text{poly}\log(n)$ operations (see Exercise 12.4 in [vzGG99]), and a different algorithm by Bernstein [Ber98] that is faster in small characteristic. However, this special case is not useful for polynomial factorization (and other applications), because in these applications $h(X)$ ends up being the input polynomial, and modular composition is used as a means of determining its (initially unknown) structure.

1.2 From modular composition to multivariate multipoint evaluation

While the algorithms of [BK78] and [HP98] reduce MODULAR COMPOSITION to matrix multiplication, in this paper, we reduce MODULAR COMPOSITION to the problem of MULTIVARIATE MULTIPOINT EVALUATION of polynomials over a ring R: given an m-variate polynomial $f(X_0, \dots, X_{m-1})$ over R of degree at most $d-1$ in each variable, and given $\alpha_i \in R^m$ for $i = 0, \dots, N-1$, compute $f(\alpha_i)$ for $i = 0, \dots, N-1$. Using this reduction, an algorithm for MULTIVARIATE MULTIPOINT EVALUATION that is optimal up to lower order terms yields an algorithm for MODULAR COMPOSITION that is optimal up to lower order terms.

In fact, we consider a slight generalization of modular composition, in which we are given a *multivariate polynomial* $f(X_1, X_2, \dots, X_m) \in R[X_1, X_2, \dots, X_m]$ and m univariate polynomials

$$g_1(X), \dots, g_m(X) \in R[X]$$

together with the modulus $h(X) \in R[X]$ (with invertible leading coefficient) and we wish to compute

$$f(g_1(X), \dots, g_m(X)) \bmod h(X).$$

We show that MULTIVARIATE MULTIPOINT EVALUATION and this general version of MODULAR COMPOSITION are in a precise sense *equivalent* (via reductions in both directions). This suggests that the reduction to MULTIVARIATE MULTIPOINT EVALUATION is the "right" approach, and indeed that progress on MODULAR COMPOSITION cannot be achieved without progress on MULTIVARIATE MULTIPOINT EVALUATION.

Recall that one can evaluate a degree n *univariate* polynomial at n evaluation points in $O(n \log^2 n)$ operations, for an amortized cost of only $O(\log^2 n)$ operations per evaluation. However, nothing similar is known for multipoint evaluations of multivariate polynomials, which seems to be a significantly more challenging problem. The only improvement over the straightforward algorithm is by Nüsken & Ziegler [NZ04], who show how to evaluate bivariate polynomials with individual degrees d at d^2 points in $O(d^{\omega_2/2+1})$ operations; their algorithm generalizes to the m-variate case where it takes $O(d^{(\omega_2/2)(m-1)+1})$ operations. Unfortunately, this is not enough to yield an improved algorithm for MODULAR COMPOSITION via the above equivalence.

1.3 Our results

In this paper, we essentially solve the MODULAR COMPOSITION problem completely, presenting algorithms that work over any finite field, whose running times are optimal up to lower order terms. We do this via the aforementioned reduction, by giving new algorithms for MULTIVARIATE MULTIPOINT EVALUATION with running times that are optimal up to lower order terms.

We give two very different algorithms for MULTIVARIATE MULTIPOINT EVALUATION. The first works over any finite field (and even more general rings of the form $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$, where E is some monic polynomial). It solves the problem by lifting to characteristic 0 followed by recursive multimodular reduction and a small number of multidimensional FFTs. A major advantage of this algorithm is that it is simple and implementable. A minor disadvantage is that it is nonalgebraic — it requires bit operations to compute the modular reductions. A purely algebraic algorithm carries some aesthetic appeal, and could be important in settings where one is working in an arithmetic model of computation (see, e.g., the pseudorandom generator of [KI04] for an example involving polynomial factorization). Our second algorithm has the advantage of being algebraic, but it works only in fields of small characteristic. It solves the problem by reducing MULTIVARIATE MULTIPOINT EVALUATION to multipoint evaluation of a *univariate* polynomial *over an extension ring*; to actually make this natural idea work requires a fairly intricate lifting using the p-power Frobenius, where p is the characteristic.

An important feature of both of our algorithms is that they do *not* rely on fast matrix multiplication. The main operations are standard fast univariate polynomial arithmetic operations, and multipoint evaluation and interpolation of univariate polynomials. All of these problems have algorithms that are asymptotically optimal up to lower order terms, and that are very reasonable in practice. In all of the settings we have mentioned where modular composition is the crucial subroutine, the other parts of the algorithms are again these standard fast and practical operations, so the algorithms derived from our new algorithm could be feasible in practice. However, we have not attempted to optimize for feasibility in our choices of parameters, so some care may be needed in order to obtain usable implementations.

In the next two subsections, we describe in more detail the techniques used in each of our two algorithms.

1.4 Techniques used in the multimodular reduction algorithm

We describe the main idea assuming the ring is $\mathbb{F}_p$, for p prime; the reduction from the general case to this case uses similar ideas.

A basic observation when considering algorithms for MULTIVARIATE MULTIPOINT EVALUATION is that *if* the evaluation points happen to be all of $\mathbb{F}_p^m$, then they can be computed all at once via the multidimensional (finite field) FFT, with an operation count that is best-possible up to logarithmic factors. More generally, if the evaluation points happen to be *well-structured* in the sense of being all of S^m for some subset $S \subseteq \mathbb{F}_p$, then by viewing $\mathbb{F}_p[X_0, X_1, \ldots, X_{m-1}]$ as $\mathbb{F}_p[X_0, X_1, \ldots, X_{m-2}][X_{m-1}]$ and applying an algorithm for univariate multipoint evaluation, and repeating m times, one can again achieve an optimal algorithm up to logarithmic factors. But these are both very special cases, and the general difficulty with MULTIVARIATE MULTIPOINT EVALUATION is contending with highly unstructured sets of evaluation points in $\mathbb{F}_p^m$.

Our main idea is to use *multimodular reduction* to transform an arbitrary set of evaluation points into a "structured" one to which the FFT solution can be applied directly. Given a m-variate polynomial f with individual degrees at most $d - 1$, and evaluation points $\alpha_i \in \mathbb{F}_p^m$, we lift f and each evaluation point to the integers by identifying the field $\mathbb{F}_p$ with the set $\{0, \ldots, p - 1\}$. We can then compute the multipoint evaluation by doing so over $\mathbb{Z}$ and reducing modulo p. To actually compute the evaluation over

$\mathbb{Z}$, we reduce modulo several smaller primes $p_1, \ldots, p_k$, producing separate instances of MULTIVARIATE MULTIPOINT EVALUATION over $\mathbb{F}_{p_i}$ for $i = 1, \ldots, k$. After solving these instances, we reconstruct the original evaluations using the Chinese Remainder Theorem.

This multimodular reduction can be applied recursively, with the primes in each round shrinking until they reach $p^* \approx (md)$ in the limit. By the last round, the evaluation points have been "packed" so tightly into each domain $\mathbb{F}_{p^*}^m$ that we can apply the multidimensional FFT to obtain *all* evaluations in $\mathbb{F}_{p^*}^m$ with little loss: d^m operations are required just to read the input polynomial f, and the FFT part of our algorithm requires only about $(dm)^m$ operations (and we will always require $m < d^{o(1)}$).

To obtain our most general result, we may need to apply three rounds of multimodular reduction; for the application to MODULAR COMPOSITION, only two rounds are needed, making the algorithm quite practical.

It is worth noting that we benefit from multimodular reduction for a quite different reason than other algorithms that employ this technique. Typically, multimodular reduction is used to reduce the "word size", when computing with large word sizes would be prohibitive or spoil the target complexity. In our case we are perfectly happy computing with word size $\log q$, so the multimodular reduction provides no benefit there. What it does do, however, is "pack" the evaluation points into a smaller and smaller space, and it does so extremely efficiently (requiring only local computations on each point). Thus, we are benefiting from the aggregate effect of applying multimodular reduction to an entire *set*, rather than directly from the reduced word size.

Our algorithm can also be used in the univariate ($m = 1$) case (via a simple transformation to the $m \gg 1$ case via the map in Definition 2.3). The overall algorithm requires only elementary modular arithmetic in $\mathbb{Z}$, and the FFT. Thus, our algorithm may be competitive, in simplicity and speed, with the "classical" algorithm for univariate multipoint evaluation (see any standard textbook, e.g., [vzGG99]). One striking contrast with the classical algorithm is that after a preprocessing step we can achieve $\mathrm{poly}(\log n, \log q)$ *actual* time for each evaluation (as opposed to amortized time); this can be interpreted as giving a powerful data structure supporting polynomial evaluation queries. This observation is fleshed out in Section 5.

1.5 Techniques used in the algebraic algorithm for small characteristic

As mentioned above, our algebraic algorithm for MULTIVARIATE MULTIPOINT EVALUATION utilizes the very natural idea of reducing to multipoint evaluation of a *univariate* polynomial *over an extension ring*. Suppose we have a multivariate polynomial $f(X_0, X_1, \ldots, X_{m-1})$ with individual degrees $d - 1$, with coefficients in $\mathbb{F}_q$. A related univariate polynomial f^* is obtained by the *Kronecker substitution*:

$$f^*(Z) = f(Z, Z^d, Z^{d^2}, \ldots, Z^{d^{m-1}}).$$

A tempting approach is to describe some (efficiently computable) mapping from evaluation points $\alpha \in \mathbb{F}_q^m$ intended for f to evaluation points $\bar{\alpha}$ in an extension field, intended for f^*, with the property that $f(\alpha)$ can be easily recovered from $f^*(\bar{\alpha})$. Then we could perform multipoint evaluation of f by mapping all of the evaluation points to their counterparts in the extension field, and then invoking a fast *univariate* multipoint evaluation algorithm to evaluate f^* at these points.

We are able to make something very close to this strategy work. To do so we need to (1) define f^* by raising to successive powers of a parameter $h \approx dm^2$ instead of d, (2) carefully construct the extension field, and (3) arrange for h to be a power of the characteristic (this is why we need small characteristic) so that we can exploit properties of the Frobenius endomorphism.

A technical requirement of our algorithm is that it needs an element of multiplicative order $h - 1$ in $\mathbb{F}_q$. If $\mathbb{F}_q$ does not contain the subfield $\mathbb{F}_h$, such an element does not even exist. As a result, we need to first extend $\mathbb{F}_q$ to guarantee such an element. This complication is not needed in settings where an order-$(h-1)$ element is already available.

The inspiration for this algorithm is two recent works in coding theory: a new variant of Reed-Solomon codes discovered by Parvaresh & Vardy [PV05] and a particular instantiation of these codes used by Guruswami & Rudra [GR06]. The analysis of the decoding algorithm in [PV05] uses the Kronecker substitution to obtain a univariate polynomial from a multivariate polynomial that carries information about the received word. This univariate polynomial is then viewed over an extension field, just as in this work. In [GR06], they utilize a particular extension field with the property that raising a polynomial (which is a canonical representative of a residue class in the extension field) to a Frobenius power is the same as shifting the polynomial by a generator of the field. We use the same trick to "store" the coordinates of an intended evaluation point in a single extension ring element, and then "access" them by raising to successive Frobenius powers.

1.6 Obtaining algorithms for transposed modular composition

The *transpose* of the modular composition problem is called MODULAR POWER PROJECTION, and it is also useful in algorithms for computing with polynomials. There is a general method (the "transposition principle") for transforming algebraic algorithms into algorithms for the transposed problem with nearly identical complexity. Our algebraic algorithm for MODULAR COMPOSITION thus immediately yields algorithms for MODULAR POWER PROJECTION with comparable operation counts, but only over fields of small characteristic.

Because our multimodular reduction-based algorithm for MODULAR COMPOSITION is nonalgebraic, the transposition principle does not directly apply. However, in Section 7.2 we show that this disadvantage can be overcome — the nonalgebraic parts of our algorithm interact well with the transposition principle — and consequently we obtain from it an algorithm for MODULAR POWER PROJECTION in any characteristic, whose running time is optimal up to lower order terms.

1.7 Application to polynomial factorization

As noted above, MODULAR COMPOSITION is used as a black box in a number of important algorithms for polynomials over finite fields, and the same is true for the transposed problem MODULAR POWER PROJECTION discussed in the previous subsection. Perhaps the most important example is factorization of degree n univariate polynomials; in this section we summarize our improvements for that problem[1].

Kaltofen & Shoup [KS98] show that an algorithm for modular composition of degree n polynomials over $\mathbb{F}_q$ requiring $C(n,q)$ bit operations gives rise to an algorithm for polynomial factorization requiring

$$n^{0.5+o(1)}C(n,q) + n^{1+o(1)}\log^{2+o(1)} q$$

bit operations. We work out this dependence on $C(n,q)$ explicitly in Section 8. Using our algorithm for modular composition, we thus obtain an algorithm for polynomial factorization requiring

$$(n^{1.5+o(1)} + n^{1+o(1)}\log q)\log^{1+o(1)} q$$

bit operations. By contrast, the best previous algorithms that work over arbitrary finite fields (von zur Gathen & Shoup [vzGS92] and Kaltofen & Shoup [KS98]) require $(n^{2+o(1)} + n^{1+o(1)}\log q)\log^{1+o(1)} q$

[1]Here we discuss our most general improvements (i.e., in arbitrary characteristic) using the nonalgebraic multimodular reduction-based algorithm. The running times therefore count bit operations, so the reader familiar with the accounting in previous work, which counts arithmetic operations in the field, should expect to see an "extra" $\log q$ factor.

and $n^{1.815}\log^{2+o(1)} q$ bit operations, respectively; we thus obtain an asymptotic improvement in the range $\log q < n$.

In Section 8 we also discuss additional problems for which our results lead to faster algorithms, including two fundamental ones: irreducibility testing, and computing minimal polynomials.

1.8 Outline

In Section 2, we give some preliminary definitions and conventions, and formally define the modular composition and multipoint evaluation problem for multivariate polynomials. In Section 3, we give the reductions showing that these two problems are essentially equivalent. In Section 4, we give our new multimodular reduction-based algorithm for multipoint evaluation of multivariate polynomials. In Section 5, we describe the data structure for polynomial evaluation arising from this algorithm. In Section 6, we give our new algebraic algorithm for multipoint evaluation of multivariate polynomials in small characteristic. In Section 7, we describe nearly-linear time algorithms for modular composition, and for its transpose (modular power projection). In Section 8, we describe some applications of our new algorithms, most notably to factorization of polynomials over finite fields. In Section 9, we mention some remaining open problems.

2 Preliminaries

In this paper, R is an arbitrary commutative ring, unless otherwise specified. For cleaner statements, we sometimes omit floors and ceilings when dealing with them would be routine. We use $o(1)$ frequently in exponents. We will always write things so that the exponentiated quantity is an expression in a single variable x, and it is then understood that the $o(1)$ term is a quantity that goes to zero as x goes to infinity.

2.1 Problem statements

The problems we are interested in are formally defined below:

Problem 2.1 (MULTIVARIATE MULTIPOINT EVALUATION). *Given $f(X_0, \ldots, X_{m-1})$ in $R[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d-1$, and evaluation points $\alpha_0, \ldots, \alpha_{N-1}$ in R^m, output $f(\alpha_i)$ for $i = 0, 1, 2, \ldots, N-1$.*

Note that the input is specified by $d^m + mN$ ring elements. The straightforward algorithm takes $\Omega(d^m N)$ ring operations, while one may hope instead for an algorithm that uses only $O(d^m + mN)$ ring operations.

Problem 2.2 (MODULAR COMPOSITION). *Given $f(X_0, \ldots, X_{m-1})$ in $R[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d-1$, and polynomials $g_0(X), \ldots, g_{m-1}(X)$ and $h(X)$, all in $R[X]$ with degree at most $N-1$, and with the leading coefficient of h invertible in R, output $f(g_0(X), \ldots, g_{m-1}(X)) \bmod h(X)$.*

We note that the term "modular composition" more commonly refers to the special case of this problem in which $m = 1$ and $N = d$. Our generalization doesn't seem to make the problem significantly more difficult to handle, though; we note, for example, that when $N = d^m$ the algorithms of [BK78, HP98] can be adapted in a straightforward way to solve the general variant in $O(N^{\omega_2/2})$ operations. Similar to above, the input is specified by $d^m + (m+1)N$ ring elements, and the straightforward algorithm takes $\Omega(d^m N)$ ring operations, while one may hope for an algorithm that uses only $O(d^m + mN)$ ring operations.

For both problems, we sometimes refer to the problem "with parameters d, m, N" if we need to specify these quantities explicitly.

Operation	Input	Output	Operations
Multiplication	$f(X), g(X)$ of degree $\leq n$	$f(X) \cdot g(X)$	$M(n)$
Remainder	$f(X), g(X)$ of degree $\leq n$	$f(X) \bmod g(X)$	$O(M(n))$
GCD	$f(X), g(X)$ of degree $\leq n$	$\gcd(f(X), g(X))$	$O(M(n)\log n)$
Evaluation	$f(X)$ of degree n; $\alpha_1, \ldots, \alpha_n$	$f(\alpha_i)$, $i = 1, \ldots n$	$O(M(n)\log n)$
Interpolation	$\alpha_0, \ldots, \alpha_n, \beta_0, \ldots, \beta_n$	$f(X)$ of degree n, $f(\alpha_i) = \beta_i$	$O(M(n)\log n)$

Figure 1: Operation counts for standard operations on univariate polynomials over a commutative ring. For interpolation, we additionally require that $\alpha_i - \alpha_j$ is a unit, for $i \neq j$. The upper bound for multiplication, $M(n)$, is $O(n \log n)$ for rings that support the FFT and $O(n \log n \log\log n)$ in general [vzGG99].

2.2 Useful facts

We have already discussed the Kronecker substitution, which can be viewed as a transformation that decreases the number of variables at the expense of increasing the degree. We now define a map that is (in a sense made precise following the definition) the "inverse" of the Kronecker substitution – it increases the number of variables while decreasing the degree:

Definition 2.3. *The map $\psi_{h,\ell}$ from $R[X_0, X_1, \ldots, X_{m-1}]$ to $R[Y_{0,0}, \ldots, Y_{m-1,\ell-1}]$ is defined as follows. Given X^a, write a in base h: $a = \sum_{j\geq 0} a_j h^j$ and define the monomial*

$$M_a(Y_0, \ldots, Y_{\ell-1}) \stackrel{\text{def}}{=} Y_0^{a_0} Y_1^{a_1} \cdots Y_{\ell-1}^{a_{\ell-1}}.$$

The map $\psi_{h,\ell}$ sends X_i^a to $M_a(Y_{i,0}, \ldots, Y_{i,\ell-1})$ and extends multilinearly to $R[X_0, X_1, \ldots, X_{m-1}]$.

Note that $\psi_{h,\ell}(f)$ can be computed in linear time in the size of f, assuming f is presented explicitly by its coefficients. Also note that $\psi_{h,\ell}$ is injective on the set of polynomials with individual degrees at most $h^\ell - 1$. For such a polynomial f, if $g = \psi_{h,\ell}(f)$, then

$$f(X_0, \ldots, X_{m-1}) = g(X_0^{h^0}, X_0^{h^1}, \ldots, X_0^{h^{\ell-1}}, X_1^{h^0}, X_1^{h^1}, \ldots, X_1^{h^{\ell-1}}, \cdots, X_{m-1}^{h^0}, X_{m-1}^{h^1}, \ldots, X_{m-1}^{h^{\ell-1}}).$$

In this sense, $\psi_{h,\ell}$ is the inverse of the Kronecker substitution.

Figure 1 gives the operation counts for standard operations on univariate polynomials that we use in the remainder of the paper. See, e.g. [vzGG99]. In this paper polynomials are always represented explicitly by a list of their coefficients. We use $M(n)$ throughout the paper as the number of operations sufficient to multiply two univariate polynomials of degree n (and we assume $M(O(n)) = O(M(n))$). Thus, when we construct an extension field (or ring) by adjoining an indeterminate X and modding out by a polynomial of degree n, arithmetic operations in the extension field (or ring) take $O(M(n))$ operations in the base field, since they entail the addition or multiplication of degree $n - 1$ polynomials followed by a remainder operation involving degree $O(n)$ polynomials.

For our first algorithm we will need the following number theory fact:

Lemma 2.4. *For all integers $N \geqslant 2$, the product of the primes less than or equal to $16 \log N$ is greater than N.*

The constant 16 is not optimal; the Prime Number Theorem implies that any constant $c > 1$ can be used for N above some bound depending on c.

Proof. The exponent of the prime p in the factorization of $n!$ equals $\sum_{i=1}^{\infty}\lfloor\frac{n}{p^i}\rfloor$ since this counts multiples of p, multiples of p^2, etc., in $\{1,\ldots,n\}$. This implies Kummer's formula

$$\binom{n}{m} = \prod_{p\leqslant n} p^{e_p}, \qquad e_p = \sum_{i=1}^{\infty}\left(\left\lfloor\frac{n}{p^i}\right\rfloor - \left\lfloor\frac{m}{p^i}\right\rfloor - \left\lfloor\frac{n-m}{p^i}\right\rfloor\right).$$

Note that $e_p \leq 1$ for $\sqrt{n} < p \leqslant n$, and $e_p \leqslant \log_p n$ for all p. From this, and the fact that $\binom{n}{\lfloor n/2\rfloor} \geqslant \binom{n}{m}$ for all m, it follows that

$$\frac{2^n}{n+1} \leqslant \binom{n}{\lfloor n/2\rfloor} \leqslant \left(\prod_{\sqrt{n}<p\leqslant n} p\right) n^{\sqrt{n}} \leqslant \left(\prod_{p\leqslant n} p\right) n^{\sqrt{n}}.$$

For $N \geqslant 50$, we have $2^n n^{-\sqrt{n}}/(n+1) \geqslant N$ for $n = \lfloor 16\log N\rfloor$, so the claim follows. For $N < 50$, the claim may be checked by hand. □

3 The reductions

In this section we give the reductions showing (essentially) that MULTIVARIATE MULTIPOINT EVALUATION and MODULAR COMPOSITION are equivalent. The reductions are not difficult, even though it appears that at least one direction of this equivalence – the one needed for our main result – was not known before[2]. The other direction, reducing multipoint evaluation of multivariate polynomials to modular composition, is just beneath the surface of the results in [NZ04].

We first reduce MODULAR COMPOSITION to MULTIVARIATE MULTIPOINT EVALUATION (this is the direction that we use in order to give our improved algorithm for MODULAR COMPOSITION).

Theorem 3.1. *Given $f(X_0,\ldots,X_{m-1})$ in $R[X_0,\ldots,X_{m-1}]$ with individual degrees at most $d-1$, and polynomials $g_0(X),\ldots,g_{m-1}(X)$ and $h(X)$, all in $R[X]$ with degree at most $N-1$, and with the leading coefficient of h invertible in R, there is, for every $2 \leq d_0 < d$, an algorithm that outputs*

$$f(g_0(X),\ldots,g_{m-1}(X)) \bmod h(X)$$

in

$$O((d^m + mN)d_0)\cdot \operatorname{poly}\log(d^m + mN)$$

ring operations plus one invocation of MULTIVARIATE MULTIPOINT EVALUATION *with parameters $d_0, m' = \ell m, N' = Nm\ell d_0$, where $\ell = \lceil\log_{d_0} d\rceil$, provided that the algorithm is supplied with N' distinct elements of R whose differences are units in R.*

Proof. We perform the following steps:

1. Compute $f' = \psi_{d_0,\ell}(f)$.

2. Compute $g_{i,j}(X) \overset{\text{def}}{=} g_i(X)^{d_0^j} \bmod h(X)$ for all i, and $j = 0,1,\ldots,\ell-1$.

[2]However, we recently learned that in an unpublished 1992 manuscript, Shoup and Smolensky used essentially the same transformation for the purpose of solving MODULAR COMPOSITION in smaller space than [BK78].

3. Select N' distinct elements of R, $\beta_0, \ldots, \beta_{N'-1}$, whose differences are units in R. Compute $\alpha_{i,j,k} \overset{\text{def}}{=} g_{i,j}(\beta_k)$ for all i, j, k using fast (univariate) multipoint evaluation.

4. Compute $f'(\alpha_{0,0,k}, \ldots, \alpha_{m-1,\ell-1,k})$ for $k = 0, \ldots, N' - 1$.

5. Interpolate to recover $f'(g_{0,0}(X), \ldots, g_{m-1,\ell-1}(X))$ (which is a univariate polynomial of degree less than N') from these evaluations.

6. Output the result modulo $h(X)$.

Correctness follows from the observation that

$$f'(g_{0,0}(X), \ldots, g_{m-1,\ell-1}(X)) \equiv f(g_0(X), \ldots, g_{m-1}(X)) \pmod{h(X)}.$$

One of the keys in using this reduction is that the left-hand-side is of sufficiently low degree so that one can afford to compute it directly and *then* reduce modulo $h(X)$, something that is not possible for the original modular composition problem, as discussed in the introduction.

The first step takes $O(d^m)$ time. For each g_i, the second step takes $O(M(N)\log(d_0^j))$ operations to compute $g_i^{d_0^j}$ by repeated squaring, and this happens for $j = 0, 1, 2, \ldots, \ell - 1$ giving an upper bound of at most $O(M(N)\ell^2 \log d_0))$ operations to compute the required powers. This happens for each g_i for a total of $O(M(N)\ell^2 m \log d_0)$ operations.

The third step takes $O(M(N')(\log N')\ell m)$ operations using fast (univariate) multipoint evaluation. The fourth step invokes fast multivariate multipoint evaluation with parameters $d_0, \ell m, N'$. The fifth step requires $O(M(N') \log N')$ operations, and the final step requires $O(M(N'))$ operations. Note that both of the $\log N'$ terms can be removed if R supports an FFT and the β's are chosen accordingly. □

Corollary 3.2. *Fix parameters d, m, N. If for every $\delta > 0$* MULTIVARIATE MULTIPOINT EVALUATION *with parameters $d_0 = d^\delta$, $m_0 = m/\delta$, N can be solved in $O((d^m + mN)^\alpha)$ operations for some constant $\alpha > 1$, then for every $\delta > 0$,* MODULAR COMPOSITION *with parameters d, m, N can be solved in $O((d^m + mN)^{\alpha+\delta})$ operations.*

The corollary is stated with matching N parameters for simplicity; it follows easily after observing that MULTIVARIATE MULTIPOINT EVALUATION with parameters $d_0, m_0, N' > N$ can be solved with $\lceil N'/N \rceil$ invocations of MULTIVARIATE MULTIPOINT EVALUATION with parameters d_0, m_0, N.

Now, we reduce MULTIVARIATE MULTIPOINT EVALUATION to MODULAR COMPOSITION, which demonstrates the equivalence of the two problems.

Theorem 3.3. *Given $f(X_0, \ldots, X_{m-1})$ in $R[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d - 1$, and evaluation points $\alpha_0, \ldots, \alpha_{N-1}$ in R^m, there is an algorithm that outputs $f(\alpha_i)$ for $i = 0, 1, \ldots, N - 1$, in*

$$O(d^m + mN) \cdot \text{poly}\log(d^m + mN)$$

ring operations, plus one invocation of MODULAR COMPOSITION *with parameters d, m, N, provided that the algorithm is supplied with N distinct elements of R whose differences are units in R.*

Proof. We perform the following steps:

1. Select distinct elements of R, $\beta_0, \ldots, \beta_{N-1}$, whose differences are units in R. Find $g_i \in R[X]$ for which $g_i(\beta_k) = (\alpha_k)_i$ for all i, k using fast univariate polynomial interpolation.

2. Produce the univariate polynomial $h(X) \overset{\text{def}}{=} \prod_k (X - \beta_k)$, and then compute $f(g_0(X) \ldots, g_{m-1}(X))$ modulo $h(X)$.

3. Evaluate this univariate polynomial at $\beta_0, \ldots, \beta_{N-1}$ using fast (univariate) multipoint evaluation, and output these evaluations.

Correctness follows from the observation that

$$f(g_1(X), \ldots, g_m(X))(\beta_k) = f(\alpha_k)$$

and the same holds when taking the left-hand-side polynomial modulo $h(X)$ since h vanishes on the evaluation points β_k.

The first step takes $O(M(N) \log N)$ operations for each interpolation, and there are m such interpolations. The second step requires $O(M(N) \log N)$ time to compute $h(X)$, and then it invokes MODULAR COMPOSITION with parameters d, m, N. The final step requires $O(M(N) \log N)$ operations. Note that the $\log N$ terms in the first and final step can be removed if R supports an FFT and the β's are chosen accordingly. □

Corollary 3.4. *Fix parameters d, m, N. If* MODULAR COMPOSITION *with parameters d, m, N can be solved in $O((d^m + mN)^\alpha)$ operations for some constant $\alpha > 1$, then* MULTIVARIATE MULTIPOINT EVALUATION *with parameters d, m, N can be solved in $O((d^m + mN)^\alpha)$ operations.*

4 Fast multivariate multipoint evaluation (in any characteristic)

We describe our first algorithm for MULTIVARIATE MULTIPOINT EVALUATION, first for prime fields, then for rings $\mathbb{Z}/r\mathbb{Z}$, and then for extension rings (and in particular, all finite fields).

4.1 Prime fields

For prime fields, we have a straightforward algorithm that uses fast Fourier transforms. The dependence on the field size p is quite poor, but we will remove that in our final algorithm using multimodular reductions.

Theorem 4.1. *Given an m-variate polynomial $f(X_0, \ldots, X_{m-1}) \in \mathbb{F}_p[X_0, \ldots, X_{m-1}]$ (p prime) with degree at most $d - 1$ in each variable, and $\alpha_0, \ldots, \alpha_{N-1} \in \mathbb{F}_p^m$, there exists a deterministic algorithm that outputs $f(\alpha_i)$ for $i = 0, \ldots, N - 1$ in*

$$O(m(d^m + p^m + N) \operatorname{poly}(\log p))$$

bit operations.

Proof. We perform the following steps to compute $f(\alpha_i)$ for $i = 0, \ldots, N - 1$.

1. Compute the reduction $\overline{f}$ of f modulo $X_j^p - X_j$ for $j = 0, \ldots, m - 1$.

2. Use a fast Fourier transform[3] to compute $\overline{f}(\alpha) = f(\alpha)$ for all $\alpha \in \mathbb{F}_p^m$.

3. Look up and return $f(\alpha_i)$ for $i = 0, \ldots, N - 1$.

[3]We need the *finite field* Fourier transform here, since we care about evaluations over $\mathbb{F}_p$.

In Step 1, the reductions modulo $X_j^p - X_j$ may be performed using md^m arithmetic operations in $\mathbb{F}_p$, for a total complexity of $O(md^m \operatorname{poly}(\log p))$.

In Step 2, we may perform the FFTs one variable at a time for a total time of $O(mp^m \operatorname{poly}(\log p))$. The details follow: we will give a recursive procedure for computing evaluations of an m-variate polynomial with individual degrees at most $p-1$ over all of $\mathbb{F}_p^m$, in time $m \cdot O(p^m \operatorname{poly}(\log p))$. When $m = 1$, we apply fast (univariate) multipoint evaluation at a cost of $O(p \operatorname{poly}(\log p))$. For $m > 1$, write $\overline{f}(X_0, X_1, \dots, X_{m-1})$ as $\sum_{i=0}^{p-1} X_0^i f_i(X_1, \dots, X_{m-1})$, and for each f_i, recursively compute its evaluations at all of $\mathbb{F}_p^{m-1}$ in time $(m-1) \cdot O(p^{m-1} \operatorname{poly}(\log p))$. Finally, for each $\beta \in \mathbb{F}_p^{m-1}$ evaluate the univariate polynomial $\sum_{i=0}^{p-1} X_0^i f_i(\beta)$ at all of $\mathbb{F}_p$ at a cost of $O(p \operatorname{poly}(\log p))$, again using fast (univariate) multipoint evaluation. The overall time is

$$(m-1) \cdot O(p^{m-1} \operatorname{poly}(\log p)) \cdot p + O(p \operatorname{poly}(\log p)) \cdot p^{m-1},$$

which equals $m \cdot O(p^m \operatorname{poly}(\log p))$ as claimed.

In Step 3, we look up N entries from a table of length p^m, for a total complexity of $O(mN \operatorname{poly}(\log p))$. This gives the stated complexity. □

4.2 Rings of the form $\mathbb{Z}/r\mathbb{Z}$

We now apply multimodular reduction recursively to remove the suboptimal dependence on p. Our main algorithm for rings $\mathbb{Z}/r\mathbb{Z}$ (r arbitrary) appears below. It accepts an additional parameter t (which will eventually be chosen to be a small constant) that specifies how many rounds of multimodular reduction should be applied.

```
Algorithm MULTIMODULAR($f, \alpha_0, \dots, \alpha_{N-1}, r, t$)

where $f$ is a $m$-variate polynomial $f(x_0, \dots, x_{m-1}) \in (\mathbb{Z}/r\mathbb{Z})[X_0, \dots, X_{m-1}]$ with degree at
most $d-1$ in each variable, $\alpha_0, \dots, \alpha_{N-1}$ are evaluation points in $(\mathbb{Z}/r\mathbb{Z})^m$, and $t$ is the number
of rounds.

  1. Construct the polynomial $\tilde{f}(X_0, \dots, X_{m-1}) \in \mathbb{Z}[X_0, \dots, X_{m-1}]$ from $f$ by replacing
     each coefficient with its lift in $\{0, \dots, r-1\}$ (via the natural identification of $\mathbb{Z}/r\mathbb{Z}$ with
     $\{0, 1, \dots, r-1\} \subseteq \mathbb{Z}$). For $i = 0, \dots, N-1$, construct the $m$-tuple $\tilde{\alpha}_i \in \mathbb{Z}^m$ from $\alpha_i$ by
     replacing each coordinate with its lift in $\{0, \dots, r-1\}$.

  2. Compute the primes $p_1, \dots, p_k$ less than or equal to $\ell = 16 \log(d^m (r-1)^{md})$, and note that
     $k \leqslant \ell$.

  3. For $h = 1, \dots, k$, compute the reduction $f_h \in \mathbb{F}_{p_h}[X_0, \dots, X_{m-1}]$ of $\tilde{f}$ modulo $p_h$. For
     $h = 1, \dots, k$ and $i = 0, \dots, N-1$, compute the reduction $\alpha_{h,i} \in \mathbb{F}_{p_h}^m$ of $\tilde{\alpha}_i$ modulo $p_h$.

  4. If $t = 1$, then for $h = 1, \dots, k$, apply Theorem 4.1 to compute $f_h(\alpha_{h,i})$ for $i = 0, \dots, N-1$;
     otherwise if $t > 1$, then run MULTIMODULAR($f_h, \alpha_{h,0}, \dots, \alpha_{h,N-1}, p_h, t-1$) to compute
     $f_h(\alpha_{h,i})$ for $i = 0, \dots, N-1$.

  5. For $i = 0, \dots, N-1$, compute the unique integer in $\{0, \dots, (p_1 p_2 \cdots p_k) - 1\}$ congruent to
     $f_h(\alpha_{h,i})$ modulo $p_h$ for $h = 1, \dots, k$, and return its reduction modulo $r$.
```

To bound the running time it will be convenient to define the function

$$\lambda_i(x) = x \log x \log\log x \log\log\log x \cdots \log^{(i-1)}(x).$$

Note that $\lambda_i(x) \leq x(\log x)^{\log^* x} = x^{1+o(1)}$ (where $\log^* x$ denotes the least nonnegative integer i such that $\log^{(i)}(x) \leqslant 1$) and that $\lambda_i(x) \leq \lambda_j(x)$ for positive x and $i < j \leqslant \log^* x$.

Theorem 4.2. *Algorithm* MULTIMODULAR *returns $f(\alpha_i)$ for $i = 0, 1, \ldots, N-1$, and it runs in*

$$O((\lambda_t(d)^m + N)\lambda_t(\log r)\lambda_t(d)^t \lambda_t(m)^{m+t+1}) \cdot O(\log^{(t)} r)^m \cdot \operatorname{poly}\log(md \log r)$$

bit operations.

Proof. Correctness follows from the fact that $0 \leqslant \tilde{f}(\tilde{\alpha}_i) \leqslant d^m(r-1)^{md} < p_1 \cdots p_k$ by Lemma 2.4, and Theorem 4.1.

Observe that in the i-th level of recursion, the primes p_h have magnitude at most $\ell_i = O(\lambda_i(m)\lambda_i(d)\log^{(i)} r)$. For convenience, set $\ell_0 = 1$.

At the i-th level of the recursion tree, the algorithm is invoked at most $\ell_0\ell_1\ell_2 \cdots \ell_{i-1}$ times. Each invocation incurs the following costs from the steps before and after the recursive call in Step 4. Step 1 incurs complexity at most $O((d^m + mN)\log r)$ at level 1, and $O((d^m + mN)\log \ell_{i-1})$ at level $i > 1$; both quantities are bounded above by $O((d^m + mN)\ell_i)$. Step 2 incurs complexity $O(\ell_i \log \ell_i)$ using the Sieve of Eratosthenes (cf. [Sho08, §5.4]). Step 3 incurs complexity $O((d^m + mN)\ell_i \operatorname{poly}(\log \ell_i))$ by using remainder trees to compute the reductions modulo $p_1, \ldots, p_k$ all at once [Ber08, §18], [vzGG99, Theorem 10.24]. Step 5 incurs complexity $O(N\ell_i \operatorname{poly}(\log \ell_i))$ as in [Ber08, §23] or [vzGG99, Theorem 10.25]. At the last level (the t-th level) of the recursion tree when the FFT is invoked, Step 4 incurs complexity $O((d^m + \ell_t^m + N)m\ell_t \operatorname{poly}(\log \ell_t))$.

Thus, using the fact that $\operatorname{poly}\log(\ell_i) \leq \operatorname{poly}\log(md \log r)$ for all i, each invocation at level $i < t$ uses

$$O((d^m + N)m\ell_i) \cdot \operatorname{poly}\log(md \log r)$$

operations while each invocation at level t uses

$$O((d^m + \ell_t^m + N)m\ell_t) \cdot \operatorname{poly}\log(md \log r)$$

operations. There are a total of $\ell_0\ell_1\ell_2 \cdots \ell_{i-1}$ invocations at level i. The total number of operations is thus

$$\left(\ell_0\ell_1\ell_2 \cdots \ell_t \cdot O((d^m + \ell_t^m + N)m) + \sum_{i=1}^{t-1} \ell_0\ell_1\ell_2 \cdots \ell_i \cdot O((d^m + N)m)\right) \cdot \operatorname{poly}\log(md \log r)$$

and using the fact that $l_i \geq 2$ for $i > 0$, this is at most

$$\begin{aligned}
&O(\ell_0\ell_1\ell_2 \cdots \ell_t) \cdot O((d^m + \ell_t^m + N)m) \cdot \operatorname{poly}\log(md \log r) \\
&\qquad \leq O(\lambda_t(m)^t \lambda_t(d)^t \lambda_{t-1}(\log r)) \cdot O((d^m + \ell_t^m + N)m) \cdot \operatorname{poly}\log(md \log r) \\
&\qquad \leq O((\lambda_t(d)^m + N)\lambda_{t-1}(\log r)\lambda_t(d)^t \lambda_t(m)^{m+t+1}) \cdot O(\log^{(t)} r)^m \cdot \operatorname{poly}\log(md \log r)
\end{aligned}$$

operations over all t levels. The bound in the theorem statement follows. □

Plugging in parameters, we find that this yields an algorithm whose running time is optimal up to lower order terms, when $m \leq d^{o(1)}$.

Corollary 4.3. *For every constant $\delta > 0$ there is an algorithm for* MULTIVARIATE MULTIPOINT EVALUATION *over $\mathbb{Z}/r\mathbb{Z}$ with parameters d, m, N, and with running time $(d^m + N)^{1+\delta} \log^{1+o(1)} r$, for all d, m, N with d sufficiently large and $m \le d^{o(1)}$.*

Proof. Let c be a sufficiently large constant (depending on δ). We may assume $m > c$ by applying the map from Definition 2.3, if necessary, to produce an equivalent instance of MULTIVARIATE MULTIPOINT EVALUATION with a greater number of variables and smaller individual degrees (and note that the quantity d^m is invariant under this map). Now if $\log^{(3)} r < m$, then we choose $t = 3$, which gives a running time of

$$O((d^{(1+o(1))m} + N)d^3 m^{m(1+o(1))(1+4/c)}(\log r)^{1+o(1)}) \cdot O(m)^m \cdot \operatorname{poly}\log(md \log r),$$

which simplifies to the claimed bound using $m \le d^{o(1)}$. Otherwise, $\log^{(3)} r \ge m$, and we choose $t = 2$, which gives a running time of

$$O((d^{(1+o(1))m} + N)d^2 m^{m(1+o(1))(1+3/c)}(\log r)^{1+o(1)}) \cdot O(\log^{(2)} r)^{\log^{(3)} r} \cdot \operatorname{poly}\log(md \log r),$$

which simplifies to the claimed bound, using $m \le d^{o(1)}$ and $O(\log^{(2)} r)^{\log^{(3)} r} \le O(\log^{o(1)} r)$.

□

4.3 Extension rings

Using algorithm MULTIMODULAR and some additional ideas, we can handle extension rings, and in particular, all finite fields. Specifically, we will work over a ring R given as $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ where E is a monic polynomial of degree e. As usual, the elements of this ring are represented as polynomials in the indeterminate Z, of degree at most $e - 1$, and with coefficients in $\mathbb{Z}/r\mathbb{Z}$; addition and multiplication in R are performed by adding or multiplying such polynomials and reducing modulo $E(Z)$. The general strategy of our algorithm is to lift to $\mathbb{Z}[Z]$, then evaluate at $Z = M$ and reduce modulo r' for suitably large integers M, r'. (Note that some finite rings are not covered by this setup, e.g., $(\mathbb{Z}/p\mathbb{Z})[X, Y]/(X^2, XY, Y^2)$. One can treat these by lifting to a multivariate polynomial ring $\mathbb{Z}[Z_1, \ldots, Z_n]$; for simplicity, we omit further details.)

The algorithm follows:

Algorithm MULTIMODULAR-FOR-EXTENSION-RING$(f, \alpha_0, \ldots, \alpha_{N-1}, t)$

where R is a finite ring of cardinality q given as $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ for some monic polynomial $E(Z)$ of degree e, f is an m-variate polynomial $f(X_0, \ldots, X_{m-1}) \in R[X_0, \ldots, X_{m-1}]$ with degree at most $d-1$ in each variable, $\alpha_0, \ldots, \alpha_{N-1}$ are evaluation points in R^m, and $t > 0$ is the number of rounds.

Put $M = d^m(e(r-1))^{(d-1)m+1} + 1$ and $r' = M^{(e-1)dm+1}$.

1. Construct the polynomial $\tilde{f}(X_0, \ldots, X_{m-1}) \in \mathbb{Z}[Z][X_0, \ldots, X_{m-1}]$ from f by replacing each coefficient with its lift, which is a polynomial of degree at most $e-1$ with coefficients in $\{0, \ldots, r-1\}$. For $i = 0, \ldots, N-1$, construct the m-tuple $\tilde{\alpha}_i \in \mathbb{Z}[Z]^m$ from α_i by replacing each coordinate with its lift, which is a polynomial of degree at most $e-1$ with coefficients in $\{0, \ldots, r-1\}$.
2. Compute the reduction $\overline{f} \in (\mathbb{Z}/r'\mathbb{Z})[X_0, \ldots, X_{m-1}]$ of $\tilde{f}$ modulo r' and $Z - M$. For $i = 0, \ldots, N-1$, compute the reduction $\overline{\alpha}_i \in (\mathbb{Z}/r'\mathbb{Z})^m$ of $\tilde{\alpha}_i$ modulo r' and $Z - M$. Note that the reductions modulo r' don't do anything computationally, but are formally needed to apply Algorithm MULTIMODULAR, which only works over finite rings of the form $\mathbb{Z}/r\mathbb{Z}$.
3. Run MULTIMODULAR$(\overline{f}, \overline{\alpha}_0, \overline{\alpha}_1, \ldots, \overline{\alpha}_{N-1}, r', t)$ to compute $\beta_i = \overline{f}(\overline{\alpha}_i)$ for $i = 0, \ldots, N-1$.
4. For $i = 0, \ldots, N-1$, compute the unique polynomial $Q_i[Z] \in \mathbb{Z}[Z]$ of degree at most $(e-1)dm$ with coefficients in $\{0, \ldots, M-1\}$ for which $Q_i(M)$ has remainder β_i modulo $r' = M^{(e-1)dm+1}$, and return the reduction of Q_i modulo r and $E(Z)$. The coefficients of polynomial Q_i are simply the digits of $\beta_i \bmod r'$ when written as an integer in base M.

Theorem 4.4. *Algorithm* MULTIMODULAR-FOR-EXTENSION-RING *returns $f(\alpha_i)$ for $i = 0, 1, \ldots, N-1$, and it runs in*

$$O((\lambda_t(d)^m + N)\lambda_t(\log q)\lambda_t(d)^{t+2}\lambda_t(m)^{m+t+3}) \cdot O(\log^{(t-1)}(d^2m^2 \log q \log\log q))^m \cdot \operatorname{poly}\log(md\log q)$$

bit operations.

Proof. To see that the algorithm outputs $f(\alpha_i)$ for $i = 0, \ldots, N-1$, note that $\tilde{f}(\tilde{\alpha}_i) \in \mathbb{Z}[Z]$ has nonnegative coefficients and its degree is at most $(e-1)dm$. Moreover, the value at $Z = 1$ of each coordinate of $\tilde{\alpha}_i$ and each coefficient of $\tilde{f}$ is at most $e(r-1)$, so $\tilde{f}(\tilde{\alpha}_i)(1) \leqslant d^m(e(r-1))^{(d-1)m+1} = M - 1$. In particular, each coefficient of $\tilde{f}(\tilde{\alpha}_i)$ belongs to $\{0, \ldots, M-1\}$. We now see that the polynomials $\tilde{f}(\tilde{\alpha}_i), Q_i \in \mathbb{Z}[Z]$ both have degree at most $(e-1)dm$ and coefficients in $\{0, \ldots, M-1\}$, and their evaluations at $Z = M$ are congruent modulo $r' = M^{(e-1)dm+1}$. This implies that the polynomials coincide, so the reduction of Q_i modulo r and $E(Z)$ agrees with the corresponding reduction of $\tilde{f}(\tilde{\alpha}_i)$, which equals $f(\alpha_i)$.

We expect a $\log q = \log(r^e)$ term in the running time, and recall that Algorithm MULTIMODULAR is invoked over a ring of cardinality $r' = M^{(e-1)(d-1)m+1}$. We have:

$$\begin{aligned}\log r' = \log(M^{(e-1)(d-1)m+1}) &\leqslant (e-1)dm \log(d^m(e(r-1))^{(d-1)m+1}+1) \\ &\leq O(ed^2m^2(\log e + \log r)) \\ &\leq O(\log q \log\log q)d^2m^2. \qquad (4.1)\end{aligned}$$

The dominant step is step 3, whose complexity is (by Theorem 4.2)

$$O((\lambda_t(d)^m + N)\lambda_t(\log r')\lambda_t(d)^t\lambda_t(m)^{m+t+1}) \cdot O(\log^{(t)} r')^m \cdot \operatorname{poly}\log(md \log r'),$$

which, using (4.1) above, yields the stated complexity. □

Similar to Corollary 4.3, we obtain:

Corollary 4.5. *For every constant $\delta > 0$ there is an algorithm for* MULTIVARIATE MULTIPOINT EVALUATION *over any ring $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ of cardinality q with parameters d, m, N, and with running time $(d^m + N)^{1+\delta} \log^{1+o(1)} q$, for all d, m, N with d sufficiently large and $m \leq d^{o(1)}$.*

Proof. The proof is the same as the proof of Corollary 4.3, except the two cases depend on m in relation to the quantity r' appearing in the proof of Theorem 4.4. The argument in the proof of Corollary 4.3 yields the claimed running time with r' in place of q; we then use the inequality $\log r' \leq O(\log q \log\log q)d^2m^2$. □

5 A data structure for polynomial evaluation

In this section we observe that it is possible to interpret our algorithm for MULTIVARIATE MULTIPOINT EVALUATION as a data structure supporting rapid "polynomial evaluation" queries.

Consider a degree n univariate polynomial $f(X) \in \mathbb{F}_q[X]$ (and think of q as being significantly larger than n). If we store f as a list of n coefficients, then to answer a single evaluation query $\alpha \in \mathbb{F}_q$ (i.e. return the evaluation $f(\alpha)$), we need to look at all n coefficients, requiring $O(n \log q)$ bit operations. On the other hand, a *batch* of n evaluation queries $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ can be answered all at once using $O(n \log^2 n)$ $\mathbb{F}_q$-operations, using fast algorithms for univariate multipoint evaluation (cf. [vzGG99]). This is often expressed by saying that the *amortized time* for an evaluation query is $O(\log^2 n)$ $\mathbb{F}_q$-operations. Can such a result be obtained in a nonamortized setting? Certainly, if we store f as a table of its evaluations in $\mathbb{F}_q$, then a single evaluation query $\alpha \in \mathbb{F}_q$ can be trivially answered in $O(\log q)$ bit operations. However, the stored data is highly redundant; it occupies space $q \log q$, when information-theoretically $n \log q$ should suffice.

By properly interpreting our algorithm for MULTIVARIATE MULTIPOINT EVALUATION, we arrive at a data structure that achieves "the best of both worlds:" we can preprocess the n coefficients describing f in nearly-linear time, to produce a nearly-linear size data structure T from which we can answer evaluation queries in time that is polynomial in $\log n$ and $\log q$. This is a concrete benefit of our approach to multipoint evaluation even for the univariate case, as it seems impossible to obtain anything similar by a suitable re-interpretation of previously known algorithms for univariate multipoint evaluation.

For clarity we state the theorem below for univariate polynomials; a similar statement holds for m-variate multivariate polynomials with individual degrees at most $d-1$ after replacing occurrences of n with d^m.

Theorem 5.1. *Let $R = (\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ be a ring of cardinality q, and let $f(X) \in R[X]$ be a degree n polynomial. Choose any constant $\delta > 0$. For sufficiently large n, one can compute from the coefficients of f in time at most*

$$T = n^{1+\delta} \log^{1+o(1)} q$$

a data structure of size at most T with the following property: there is an algorithm that given $\alpha \in \mathbb{F}_q$, computes $f(\alpha)$, in time

$$\operatorname{poly}\log n \cdot \log^{1+o(1)} q$$

with random access to the data structure.

Proof. The first step is to apply the map $\psi_{d,m}$ from Definition 2.3 to f. At the end of this proof we will specify the parameters d, m to use (they depend only on $r, \deg(E)$, and n); for now we only need to know that $d^m = n$. We also have a parameter t that will be chosen at the end of this proof.

After applying $\psi_{d,m}$, we have an m-variate polynomial f, which we process by following the operations of MULTIMODULAR-FOR-EXTENSION-RING step-by-step, ignoring the ones that process the evaluation points. The key observation is that these computations do not depend on the evaluation points, and can thus comprise a preprocessing phase that produces the desired data structure.

We go through the steps here: Steps 1 and 2 of MULTIMODULAR-FOR-EXTENSION-RING produce $\overline{f}$ with coefficients in $\mathbb{Z}/r'\mathbb{Z}$. Step 3 calls MULTIMODULAR which apply t rounds of multimodular reduction to finally produce *reduced polynomials* $f_{s_1,s_2,\ldots,s_t} \in \mathbb{F}_{s_t}[X_0, X_1, \ldots, X_{m-1}]$ for certain sequences $s_1, s_2, \ldots, s_t$ of primes (each such sequence has s_1 among the ℓ_1 primes computed in Step 2 of the top-level invocation of MULTIMODULAR; s_2 is among the ℓ_2 primes computed in Step 2 of the next-level invocation of MULTIMODULAR with s_1 as its parameter "r", etc...). In other words, the primes s_i are the moduli in a sequence of t recursive calls to MULTIMODULAR. At the bottom level of the recursion, each reduced polynomial $f_{s_1,s_2,\ldots,s_t}$ is evaluated (in Step 4, which applies Theorem 4.1) over its *entire* domain $\mathbb{F}_{s_t}^m$ (i.e., we ignore Step 3 in the proof of Theorem 4.1, which looks up evaluation points, and instead return the table of all evaluations over $\mathbb{F}_{s_t}^m$ computed in Step 2 of that proof). Our data structure consists of these tables of evaluations of each of the reduced polynomials $f_{s_1,s_2,\ldots,s_t}$, each one labeled by the sequence $s_1, s_2, \ldots, s_t$, together with f itself.

Using notation from the proof of Theorem 4.2, there are at most $\ell_1\ell_2\cdots\ell_t$ reduced polynomials, each p_t has magnitude at most ℓ_t, and it holds that $\ell_i = O(\lambda_i(m)\lambda_i(d)\log^{(i)} r')$. Referring to the proof of Theorem 4.1, we see that the cost incurred to produce the required tables of evaluations is at most

$$\begin{aligned} T &= \ell_1\ell_2\cdots\ell_t \cdot O(m\ell_t^m) \cdot \operatorname{poly}\log(\ell_t) \\ &\leq O(\lambda_t(m)^{t+m+1}\lambda_t(d)^{t+m}\lambda_{t-1}(\log r')) \cdot (\log^{(t)} r')^m \cdot \operatorname{poly}\log(md\log r') \end{aligned}$$

At this point, an evaluation query $\alpha' \in R$ can be answered from the tables as follows. First compute the point $\alpha = (\alpha', \alpha'^d, \ldots, \alpha'^{d^{m-1}}) \in R^m$. Note that $f(\alpha)$ is our desired output (since f is the input polynomial *after* the map $\psi_{d,m}$ is applied to it). We now follow the operations of MULTIMODULAR-FOR-EXTENSION-RING on an evaluation point step-by-step, ignoring the ones that process the polynomial. As above we go through them one by one: Steps 1 and 2 of MULTIMODULAR-FOR-EXTENSION-RING produce $\overline{\alpha} \in (\mathbb{Z}/r'\mathbb{Z})^m$. Step 3 calls MULTIMODULAR which applies t rounds of multimodular reduction exactly as above to finally produce *reduced evaluation points* $\alpha_{s_1,s_2,\ldots,s_t} \in \mathbb{F}_{s_t}^m$ for certain sequences $s_1, s_2, \ldots, s_t$ of primes. The evaluations $f_{s_1,s_2,\ldots,s_t}(\alpha_{s_1,s_2,\ldots,s_t})$ can be found in the pre-computed tables that comprise the data structure for f. We reconstruct $\overline{f}(\overline{\alpha})$ by "going up the recursion tree": the evaluations in the tables supply the needed data to execute Step 5 of the bottom-level invocations of MULTIMODULAR (for the single evaluation point of interest); these in turn supply the needed data to execute Step 5 of the next-to-bottom level invocations of MULTIMODULAR, etc... Finally we obtain an evaluation β of $\overline{f}$, which is the data needed to execute Step 4 of MULTIMODULAR-FOR-EXTENSION-RING, which produces $f(\alpha)$ as desired.

The dominant cost in processing an evaluation query is the recursive reconstruction of the output from the table lookups. Again adopting the notation from the proof of Theorem 4.2, the Chinese Remainder Theorem reconstruction (in Step 5 of MULTIMODULAR) is invoked $\ell_1\ell_2\cdots\ell_{i-1}$ times at level i, each time with cost $O(\ell_i \operatorname{poly}\log(\ell_i))$. The overall cost for an evaluation query is thus dominated by

$$\begin{aligned}\sum_{i=1}^{t} \ell_1\ell_2\cdots\ell_{i-1}\cdot O(\ell_i \operatorname{poly}\log(\ell_i)) &\leq \sum_{i=1}^{t} \ell_1\ell_2\cdots\ell_i \cdot \operatorname{poly}\log(md\log r') \\ &\leq O(\ell_1\ell_2\cdots\ell_t)\cdot \operatorname{poly}\log(md\log r') \\ &\leq O(\lambda_t(m)^t\lambda_t(d)^t\lambda_{t-1}(\log r'))\cdot \operatorname{poly}\log(md\log r')\end{aligned}$$

It remains to choose the parameters d, m and t. If $r' > 2^{2^n}$, then we choose $d = n, m = 1, t = 2$; if $r' \leq 2^{2^n}$, then choose $d = \log^c n$ and $m = (\log n)/(c\log\log n)$ for a sufficiently large constant c, and $t = 4$. These choices give the claimed running times for preprocessing and queries, with r' in place of q. As in the proof of Theorem 4.4, we have $\log r' \leq O(\log q \log\log q)d^2m^2$, which completes the proof. $\square$

Theorem 5.1 is surprising in light of a number of lower bounds for this problem under certain restrictions. For example, in the purely algebraic setting, and when the underlying field in $\mathbb{R}$, Belaga [Bel61] shows a lower bound on the query complexity of $\lfloor \frac{3n}{2} \rfloor + 1$ (and Pan [Pan66] has given a nearly-matching upper bound). Miltersen [Mil95] proves that the trivial algorithm (with query complexity n) is essentially optimal when the field size is exponentially large and the data structure is limited to polynomial size, and he conjectures that this lower bound holds for smaller fields as well (this is in an algebraic model that does not permit the modular operations we employ). Finally, Gál and Miltersen [GM07] show a lower bound of $\Omega(n/\log n)$ on the product of the *additive redundancy* (in the data structure size) and the query complexity, thus exhibiting a tradeoff that rules out low query complexity when the data structure is required to be very small (i.e., significantly smaller than $2n$).

6 An algebraic algorithm in small characteristic

In this section we describe an algorithm for MULTIVARIATE MULTIPOINT EVALUATION that is completely different from the one in Section 4. The advantage of this algorithm is that it is algebraic (and it achieves an operation count that is optimal up to lower order terms); the disadvantage is that it works only over fields of small characteristic. We present our algorithm in an algebraic model with one nonstandard feature; this is discussed next, followed by the algorithm itself.

6.1 The algebraic model

Fix a field $\mathbb{F}_q$ of characteristic p. In the standard algebraic model, the basic operations are addition, subtraction, multiplication and division. A special feature of characteristic p is the existence of the (absolute) Frobenius automorphism σ_0, given by $x \mapsto x^p$, and we make heavy use of the existence of this automorphism. We include the operation of *applying* σ_0^{-1} as a basic operation in our presentation below. This is nonstandard, but we believe it is justified as follows:

- It is easy to recover a pure $\mathbb{F}_q$-algebraic algorithm by replacing each operation that applies σ_0^{-1} with the explicit computation of $x^{q/p}$ by repeated squaring (and note that $\sigma_0^{-1}(x) = x^{q/p}$). This replacement contributes at most an additional $O(\log(q/p))$ factor to the operation counts. This means that it is easy to read off an upper bound in the standard model from the operation counts in our presentation below.

- Any (division-free) algebraic computation of σ_0^{-1} in the standard model *must* entail $\log_2(q/p)$ $\mathbb{F}_q$-operations by a simple degree argument: fewer operations would give rise to a polynomial $f(X)$ of degree less than q/p, and yet $f(X)^p - X$ vanishes on all of $\mathbb{F}_q$, so it must have degree at least q. However, this cost is misleadingly high, because in practice something much more efficient is possible. In practice, $\mathbb{F}_q$ is usually represented as $\mathbb{F}_p[X]/(f(X))$ for a degree d irreducible f, and in this representation σ_0^{-1} can be applied at a cost of $O(p^{1+o(1)}d^{1+o(1)})$ basic $\mathbb{F}_p$-operations; see Theorem 6.1 below. This is nearly-linear time (like the other basic operations) provided that $p = d^{o(1)}$, as it will be in all of our algebraic algorithms below. (Alternatively, one can compute $X^{q/p} \bmod f(X)$ using our nonalgebraic algorithm for MODULAR COMPOSITION, starting with $X^p \bmod f(X)$ and then computing X^{p^i} for $i = 2^1, 2^2, \ldots$. Applying σ_0^{-1} to an element of $\mathbb{F}_q$ then entails only a single modular composition.)

For the remainder of this paper, unless otherwise noted, we assume this algebraic model when discussing algebraic algorithms.

We now give the promised algorithm for computing σ_0^{-1} in a finite field. This is a variation on a construction from [PT09].

Theorem 6.1. *There is an algorithm which, given a prime number p, an irreducible monic polynomial $f(X)$ in $\mathbb{F}_p[X]$ of degree d, and a polynomial $g(X)$ in $\mathbb{F}_p[X]$ of degree at most $d-1$, returns the unique polynomial $h(X)$ in $\mathbb{F}_p[X]$ of degree at most $d-1$ such that*

$$h(X)^p \bmod f(X) = g(X),$$

and runs in $O(p^{1+o(1)}d^{1+o(1)})$ $\mathbb{F}_p$-operations.

Proof. The existence and uniqueness of $h(X)$ follows from the fact that the absolute Frobenius map on the finite field $\mathbb{F}_p[X]/(f(X))$ is a bijection. Since $h(X)^p = h(X^p)$ and $f(X)^p = f(X^p)$, the congruence

$$h(X)^p \equiv g(X) \pmod{f(X)}$$

is equivalent to

$$h(X^p)f(X)^{p-1} \equiv g(X)f(X)^{p-1} \pmod{f(X^p)}. \tag{6.1}$$

Write

$$\begin{aligned} f(X)^{p-1} &= A_0(X^p) + A_1(X^p)X + \cdots + A_{p-1}(X^p)X^{p-1} \\ g(X)f(X)^{p-1} &= B_0(X^p) + B_1(X^p)X + \cdots + B_{p-1}(X^p)X^{p-1} \end{aligned}$$

with $A_0, \ldots, A_{p-1}, B_0, \ldots, B_{p-1}$ in $\mathbb{F}_p[X]$. Viewing $\mathbb{F}_p[X]$ as a dimension p vectorspace over $\mathbb{F}_p[X^p]$, and noting that $X^0, X^1, \ldots, X^{p-1}$ form a basis, Eq. (6.1) forces

$$h(X)A_i(X) \equiv B_i(X) \pmod{f(X)} \qquad \text{for } i = 0, \ldots, p-1.$$

For any i such that $A_i(X)$ is not divisible by $f(X)$ (which implies that A_i and f are relatively prime, since f is irreducible), the ith congruence above, by itself, uniquely determines $h(X)$ modulo $f(X)$. It follows that we correctly compute $h(X)$ using the following algorithm.

1. Compute $f(X)^{p-1}$ and $g(X)f(X)^{p-1}$, and write them as

$$\begin{aligned} f(X)^{p-1} &= A_0(X^p) + \cdots + A_{p-1}(X^p)X^{p-1} \\ g(X)f(X)^{p-1} &= B_0(X^p) + \cdots + B_{p-1}(X^p)X^{p-1} \end{aligned}$$

   with $A_0, \ldots, A_{p-1}, B_0, \ldots, B_{p-1}$ in $\mathbb{F}_p[X]$.

2. Note that $\deg(A_i)p + i \leq \deg(f(X)^{p-1}) = d(p-1)$, which implies (for all i) that $\deg(A_i) < d$, and hence A_i cannot be divisible by $f(X)$. There is an i (namely, the i for which $i \equiv \deg(f(X)^{p-1}) = d(p-1) \pmod{p}$) such that the leading coefficient of A_i is the leading coefficient of $f(X)^{p-1}$, and hence A_i is nonzero. Select this A_i and compute the unique polynomial $h(X)$ of degree at most $d-1$ such that

$$h(X)A_i(X) \equiv B_i(X) \pmod{f(X)}.$$

   Note that the index i and the polynomial A_i depend only on f, not on g, and so can be precomputed.

Since both steps involve standard operations (multiplication, GCD, etc...) on polynomials of degree $O(pd)$, their complexities are each bounded by $O((pd)^{1+o(1)})$ operations in $\mathbb{F}_p$ (refer to Figure 1). □

6.2 The algorithm

As described in Section 1.5, our algorithm operates by reducing multipoint evaluation of the target *multivariate* polynomial f to multipoint evaluation of a related *univariate* polynomial f^* obtained by substituting h-th powers of a single variable for the m different variables of f (the "Kronecker substitution"). The given m-variate polynomial f will have coefficients in a field $\mathbb{F}_q$ and the parameter h will be a power of the characteristic. We will actually view f as a polynomial with coefficients in an extension *ring* $R = \mathbb{F}_q[W]/(P(W))$ for some polynomial P (not necessarily irreducible over $\mathbb{F}_q$). The reason for this complication is that the algorithm needs a special element η that satisfies two properties:

1. the multiplicative order of η is $h-1$, and
2. $\eta^i - \eta^j$ is invertible for all $i, j \in \{0, 1, 2, \ldots, m-1\}$, with $i \neq j$.

We will construct R so that we can easily get our hands on such a η. If an element of order $h-1$ is already available in $\mathbb{F}_q$, then it automatically satisfies the second property because $\mathbb{F}_q$ is a field, and there is no need to pass to the extension ring R.

We now describe in detail how to construct the extension ring R, and find η. Fix parameters d and m, and a field $\mathbb{F}_q$ with characteristic p. Let $h = p^c$ be the smallest integer power of p that is larger than m^2d. Construct the ring $R = \mathbb{F}_q[W]/(P(W))$, where $P(W)$ is a degree c polynomial with coefficients in $\mathbb{F}_p$, that is irreducible over $\mathbb{F}_p$[4]. Notice that $\mathbb{F}_p[W]/(P(W)) \subseteq R$ and also that $\mathbb{F}_q \subseteq R$, and that these embeddings are easy to compute. Choose η to be a primitive element of the field $\mathbb{F}_p[W]/(P(W))$. This η clearly has multiplicative order $h-1$, and because the elements η^i for $i = 0, 1, \ldots, m-1$ are distinct elements of a field, the second property above is also satisfied. Figure 2 depicts the construction of R.

Given the m-variate polynomial f over R, we want to be able to evaluate it at many points in $\mathbb{F}_q^m \subseteq R^m$. Our strategy will be to lift the evaluation points to elements of an extension ring S, evaluate a related

[4]One may wonder why we don't simply demand that P be irreducible over $\mathbb{F}_q$, so that R is a field. The reason is that we can afford to find an irreducible over $\mathbb{F}_p$, even by brute force search, while searching for an irreducible over $\mathbb{F}_q$ would introduce an undesirable dependence on q to the operation counts.

$$\begin{tikzcd}
& S = R[Z]/(E(Z)) \arrow[d, dash] & \\
& R = \mathbb{F}_q[W]/(P(W)) \arrow[dl, dash] \arrow[dr, dash] & \\
\mathbb{F}_q \arrow[dr, dash] & & \mathbb{F}_p[W]/(P(W)) \arrow[dl, dash] \\
& \mathbb{F}_p &
\end{tikzcd}$$

Figure 2: Containment diagram. Our input polynomial will be over $\mathbb{F}_q$, but we view it as a polynomial over the extension ring R. We will end up evaluating a related polynomial at elements of the further extension S.

univariate polynomial f^* at those points, and then project each resulting evaluation back to an element of R. We choose the ring S to be the extension ring $R[Z]/(E(Z))$, where $E(Z) \overset{\text{def}}{=} Z^{h-1} - \eta$. Refer to Figure 2.

Let σ be (a power of) the Frobenius endomorphism from R to R, given by $x \mapsto x^h$. The "lift" map $\phi : \mathbb{F}_q^m \to S$ is defined as follows: given $\alpha = (\alpha_0, \ldots, \alpha_{m-1}) \in \mathbb{F}_q^m \subseteq R^m$, $\phi(\alpha)$ is the (residue class whose canonical representative is the) degree $m-1$ polynomial $g_\alpha(Z) \in R[Z]$ which has

$$g_\alpha\left(\eta^i\right) = \sigma^{-i}(\alpha_i) \text{ for } i = 0, 1, 2 \ldots, m-1. \tag{6.2}$$

Note that g_α is well defined because although σ^i is only an *endomorphism* of R (under which certain elements may have no preimage), we only demand preimages of elements of $\mathbb{F}_q \subseteq R$, and σ^i is an *automorphism* when restricted to $\mathbb{F}_q$. Recall also that $m < h$ by our choice of h, and that the canonical representatives of the residue classes of S are polynomials in $R[Z]$ of degree less than $h-1$, so g_α is indeed a canonical representative as asserted above.

The "project" map $\pi : S \to R$ that recovers the evaluation of the original multivariate polynomial f from an evaluation of the univariate polynomial f^* is defined as follows: given an element of S whose canonical representative is the polynomial $g(Z) \in R[Z]$ (with degree less than $h-1$), $\pi(g)$ is the evaluation $g(1)$.

Our main lemma shows how to recover the evaluation of the m-variate polynomial f at a point $\alpha \in \mathbb{F}_q^m \subseteq R^m$, from the evaluation of the univariate polynomial f^* at an element of the extension ring S.

Lemma 6.2. *Let $f(X_0, X_1, \ldots, X_{m-1})$ be a polynomial in $\mathbb{F}_q[X_0, X_1, \ldots, X_{m-1}]$ with individual degrees $d-1$, and suppose $\mathbb{F}_q$ has characteristic p. Define h, R, E, S, ϕ, π as above, and define the univariate polynomial $f^*(Y) \in S[Y]$ by:*

$$f^*(Y) \overset{\text{def}}{=} f(Y, Y^h, Y^{h^2}, \ldots, Y^{h^{m-1}}).$$

For every $\alpha \in \mathbb{F}_q^m \subseteq R^m$, the following identity holds: $\pi(f^*(\phi(\alpha))) = f(\alpha)$.

Proof. Fix $\phi(\alpha)$, which is an element of $R[Z]/(E(Z))$. Let $g_\alpha(Z) \in R[Z]$ be its (degree $m-1$) canonical representative, and denote by $\sigma^i(g_\alpha)$ the polynomial obtained by applying σ^i to the coefficients of g_α. Then we have:

$$\begin{aligned}
(g_\alpha(Z))^{h^i} &= \sigma^i(g_\alpha)(Z^{h^i}) \\
&= \sigma^i(g_\alpha)(Z^{h^i-1}Z) \\
&\equiv \sigma^i(g_\alpha)(\eta^{(h^i-1)/(h-1)}Z) \pmod{E(Z)} \\
&= \sigma^i(g_\alpha)(\eta^{(h^{i-1}+h^{i-2}+\cdots+h^0)}Z) \\
&= \sigma^i(g_\alpha)(\eta^i Z),
\end{aligned}$$

where the last equality used the fact that η has order $h-1$ and so it is fixed under σ. For convenience, let us denote by $g_\alpha^{(i)}(Z)$ the polynomial $(g_\alpha(Z))^{h^i} \bmod E(Z)$. A crucial point that we will use shortly is that $\deg(g_\alpha^{(i)}) = \deg(g_\alpha)$. The above equation implies that

$$g_\alpha^{(i)}(1) = \sigma^i(g_\alpha)\left(\eta^i\right) = \sigma^i\left(g_\alpha\left(\sigma^{-i}\eta^i\right)\right) = \sigma^i\left(g_\alpha\left(\eta^i\right)\right) = \sigma^i(\sigma^{-i}\alpha_i) = \alpha_i, \tag{6.3}$$

where the third equality again used the fact that η is fixed under σ, and the fourth equality used Eq. (6.2).

When we evaluate the polynomial f^* at the element of S whose canonical representative is g_α we get the element of S whose canonical representative is:

$$f(g_\alpha^{(0)}(Z), g_\alpha^{(1)}(Z), \ldots, g_\alpha^{(m-1)}(Z)) \bmod E(Z).$$

Now f is a polynomial with total degree at most dm, and each $g_\alpha^{(i)}$ is a polynomial of degree at most $m-1$. Therefore, since E has degree at least $dm^2 > dm(m-1)$, this polynomial is just

$$f(g_\alpha^{(0)}(Z), g_\alpha^{(1)}(Z), \ldots, g_\alpha^{(m-1)}(Z)),$$

and evaluating at 1 gives (using Eq. (6.3)):

$$f(g_\alpha^{(0)}(1), g_\alpha^{(1)}(1), \ldots, g_\alpha^{(m-1)}(1)) = f(\alpha_0, \alpha_1, \ldots, \alpha_{m-1})$$

as claimed. □

The next theorem applies the strategy we have developed above to the MULTIVARIATE MULTIPOINT EVALUATION problem. Note that this algorithm requires a field (as opposed to the more general rings handled by the algorithm of Section 4) and that an optimal dependence of the operation count on the input size $(N + d^m)$ (up to lower order terms) can only be achieved when the characteristic p is at most $d^{o(1)}$.

Theorem 6.3. *Given $f(X_0, \ldots, X_{m-1})$ in $\mathbb{F}_q[X_0, \ldots, X_{m-1}]$ with individual degrees at most $d-1$, and evaluation points $\alpha_0, \ldots, \alpha_{N-1}$ in $\mathbb{F}_q^m$, there is an algorithm that outputs $f(\alpha_i)$ for $i = 0, 1, 2, \ldots, N-1$, in*

$$O((N + d^m)(m^2 p)^m) \cdot \mathrm{poly}(d, m, p, \log N)$$

field operations.

Proof. We perform the following steps:

1. Choose $h = p^c$ to be the smallest power of p that is at least $m^2 d$. Find a degree c irreducible polynomial $P(W)$ over $\mathbb{F}_p$, and a primitive element η of $\mathbb{F}_p[W]/(P(W))$. Define the ring $R = \mathbb{F}_q[W]/(P(W))$, and the ring $S = R[Z]/(E(Z))$, where $E(Z) = Z^{h-1} - \eta$, as above.

2. For $i = 0, 1, 2, \ldots, N-1$, compute the canonical representative of $\phi(\alpha_i)$: the degree $m-1$ polynomial $g_{\alpha_i}(Z) \in R[Z]$.
3. Produce the univariate polynomial $f^*(Y) = f(Y, Y^h, Y^{h^2}, \ldots, Y^{h^{m-1}})$ over S.
4. Evaluate f^* at the points $g_{\alpha_i}(Z)$, and for each evaluation apply π to recover $f(\alpha_i)$.

Step 1 requires constructing the field $\mathbb{F}_h$ and finding a primitive element. This can be done by brute force in $\mathrm{poly}(h)$ operations, although much better algorithms are available.

Each polynomial g_{α_i} computed in Step 2 requires the following operations (recall Eq. (6.2)): first, we need to compute $\sigma^{-j}(\alpha_i)_j$ for $j = 0, 1, \ldots, m-1$. This is done[5] by applying σ_0^{-1} cj times to $(\alpha_i)_j$ (recall that $\sigma = \sigma_0^c$), and note that this is the only place we use this nonstandard basic operation. The overall cost of doing this for all i and j is $O(Nm^2c)$. Next, we perform N polynomial interpolations in R, each costing $O(M(m)\log m)$ operations in R, or $O(M(m)\log mM(c))$ operations in $\mathbb{F}_q$. Note that for every two interpolation points η^i, η^j, the difference $\eta^i - \eta^j$ is a unit in R (since η is an element of $\mathbb{F}_p[W]/(P(W))$ which is a field). This is required for the interpolation step. The total cost for Step 2 is

$$O(N(m^2 \log h + M(m)(\log m)M(c)))$$

$\mathbb{F}_q$-operations.

Step 4 is a univariate multipoint evaluation problem. We have N elements of S, and a univariate polynomial f^* over S, of degree at most dmh^m. If $L = \max(N, dmh^m)$, this step requires $O(M(L)\log L)$ operations in S, or

$$O(M(L)(\log L)M(h)M(c))$$

$\mathbb{F}_q$-operations, using fast univariate multipoint evaluation. The N applications of π take $O(Nh)$ operations in R, or $O(NhM(c))$ $\mathbb{F}_q$-operations.

The final complexity estimate is obtained by using the bound $h \leqslant pm^2d$. (The dominant step is Step 4.) $\square$

Corollary 6.4. *For every constant $\delta > 0$ there is an algorithm for* MULTIVARIATE MULTIPOINT EVALUATION *over $\mathbb{F}_q$ with parameters d, m, N, and with operation count $(d^m + N)^{1+\delta}$, for all d, m, N with d, N sufficiently large, provided $m \leq d^{o(1)}$ and the characteristic $p \leq d^{o(1)}$.*

Proof. Let c be a sufficiently large constant (depending on δ). We may assume $m > c$ by applying the map from Definition 2.3, if necessary, to produce an equivalent instance of MULTIVARIATE MULTIPOINT EVALUATION with more variables and smaller individual degrees (and note that the quantity d^m is invariant under this map). The operation count of Theorem 6.3 has an "extra" multiplicative factor of $(m^2p)^m \cdot \mathrm{poly}(d, p, m, \log N)$, and we claim this can be made to be at most $(N + d^m)^\delta$. This is because $m \leq d^{o(1)}$ (so $m^{2m+O(1)} \leq d^{o(m)}$), and $p \leq d^{o(1)}$ (so $p^{m+O(1)} \leq d^{o(m)}$), and $d^{O(1)} \leq (d^m)^{O(1)/c}$ (recall we are choosing c sufficiently large), and finally $\mathrm{poly}\log N \leq N^\delta$ for sufficiently large N. $\square$

7 Fast modular composition, and its transpose

We now obtain fast algorithms for MODULAR COMPOSITION and its transpose, MODULAR POWER PROJECTION, via the reduction of Theorem 3.1, and the transposition principle.

[5]The conference paper [Uma08] erroneously claimed at this point that the computation of $\sigma^{-j}(\alpha_i)_j$ could be performed using $O(m \log h)$ standard $\mathbb{F}_q$-operations. Here we recover essentially the same theorem statement as Theorem 4.2 in [Uma08], but only in the nonstandard algebraic model; in the standard model the operation count requires an extra $O(\log(q/p))$ factor, as discussed in Section 6.1.

7.1 Modular composition

By applying the reduction in Theorem 3.1, we obtain a nearly-linear time algorithm for MODULAR COMPOSITION.

Theorem 7.1. *Let R be a finite ring of cardinality q given as $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ for some monic polynomial $E(Z)$. For every constant $\delta > 0$, if we have access to Nd^{δ} distinct elements of R whose differences are units in R, there is an algorithm for* MODULAR COMPOSITION *over R with parameters d, m, N, and with running time $(d^m + N)^{1+\delta} \log^{1+o(1)} q$, for all d, m, N with d, N sufficiently large, provided $m \leq d^{o(1)}$. If R is isomorphic to the field $\mathbb{F}_q$ with characteristic $p \leq d^{o(1)}$, then the algorithm can be taken to be algebraic, with operation count $(d^m + N)^{1+\delta}$.*

Proof. Let c be a a sufficiently large constant (depending on δ), and set $d_0 = d^{1/c}$ and $m_0 = cm$. Then applying Theorem 3.1, we obtain an algorithm for MODULAR COMPOSITION requiring $O((d^m + mN)d_0) \cdot \operatorname{poly}\log(d^m + mN)$ operations plus one invocation of MULTIVARIATE MULTIPOINT EVALUATION with parameters $d_0, m_0, N' = Nmcd_0$. By our choice of c, and the fact that $m \leq d^{o(1)}$ and d, N are sufficiently large, this operation count is at most $(d^m + N)^{1+\delta}$. By Corollary 4.5, the instance of MULTIVARIATE MULTIPOINT EVALUATION can be solved in time $(d^m + N)^{1+\delta} \log^{1+o(1)} q$, or with $(d^m + N)^{1+\delta}$ field operations via Corollary 6.4 if we are working over the field $\mathbb{F}_q$ with characteristic $p \leq d^{o(1)}$. □

We remark that for the "standard" parameter setting of $m = 1$ and $N = d$, one can achieve the claimed running time by taking $t = 2$ when solving the MULTIVARIATE MULTIPOINT EVALUATION instance via Algorithm MULTIMODULAR-FOR-EXTENSION-RING. This makes the overall algorithm (arguably) practical and implementable. Indeed, use of a single round of multimodular reduction is quite common in practice; for instance, Shoup's NTL library [Sho] uses multimodular reduction for most basic arithmetic involving multiprecision integer polynomials.

The following corollary addresses the most common special case of Theorem 7.1:

Corollary 7.2. *For every $\delta > 0$, there is an algorithm for* MODULAR COMPOSITION *over $\mathbb{F}_q$ with parameters $d, m = 1, N = d$ running in $d^{1+\delta} \log^{1+o(1)} q$ bit operations, for sufficiently large d. If the characteristic p is at most $d^{o(1)}$, then the algorithm may be taken to be algebraic, with operation count $d^{1+\delta}$.*

Proof. Construct an extension field $\mathbb{F}_{q'}$ of $\mathbb{F}_q$ with cardinality at least $d^{1+\delta}$, then apply Theorem 7.1 with $R = \mathbb{F}_{q'}$. □

Remark. In the running times claimed in Corollaries 4.3, 4.5, 6.4, 7.2, and Theorem 7.1, we have chosen to present bounds that interpret "almost linear in x" as meaning "for all $\delta > 0$, there is an algorithm running in time $x^{1+\delta}$ for sufficiently large x." In all cases, it is possible to choose δ to be a subconstant function of the other parameters, giving stronger, but messier, bounds.

7.2 Fast modular power projection

In this section, we restrict ourselves to "standard" parameter setting for MODULAR COMPOSITION— in which $m = 1$ and $N = d$. We consider the "transpose" of MODULAR COMPOSITION, defined next:

Problem 7.3 (MODULAR POWER PROJECTION). *Given a linear form $\tau : R^d \to R$, and polynomials $g(X), h(X)$ in $R[X]$, each with degree at most $d - 1$, and with the leading coefficient of h a unit in R, output $\tau(g(X)^i \bmod h(X))$ for $i = 0, 1, \ldots, d - 1$. Here we identify a polynomial with the vector of its coefficients.*

One can view MODULAR COMPOSITION as multiplying the $d \times 1$ column vector of coefficients of f on the left by the $d \times d$ matrix $A_{g,h}$, whose columns are the coefficients of $g(X)^i \bmod h(X)$ for $i = 0, 1, \ldots, d-1$. Then MODULAR POWER PROJECTION is the problem of multiplying the column vector of coefficients of τ on the left by the transpose of $A_{g,h}$.

By a general argument (the "transposition principle"), *linear* straight-line programs computing a linear map yield linear straight-line programs with essentially the same complexity for computing the transposed map.

Theorem 7.4 ([BCS97, Thm. 13.20])**.** *Let $\phi : R^n \to R^m$ be a linear map that can be computed by a linear straight-line program of length L and whose matrix in the canonical basis has z_0 zero rows and z_1 zero columns. Then the transposed map $\phi^t : R^m \to R^n$ can be computed by a linear straight-line program of size $L - n + m - z_0 + z_1$.*

One can verify that the algebraic algorithm of Corollary 7.2 (which may be used in the small characteristic case), when written as a straight-line program, computes only linear forms in the coefficients of the input polynomial f. This is because the computations involving the input polynomials g and h, including all applications of the nonstandard algebraic operation σ_0^{-1}, can be isolated into a preprocessing phase, which does not involve f. Thus Theorem 7.4 applies, and immediately gives:

Theorem 7.5. *For every $\delta > 0$, there is an algebraic algorithm for* MODULAR POWER PROJECTION *over $\mathbb{F}_q$ with operation count $d^{1+\delta}$, for sufficiently large d, provided the characteristic p is at most $d^{o(1)}$.*

Unfortunately the general-characteristic algorithm of Corollary 7.2 (i.e., Algorithm MULTIMODULAR-FOR-EXTENSION-RING) does *not* compute only linear forms in the coefficients of polynomial f (because of the lifting to characteristic 0 followed by modular reduction) so we cannot apply Theorem 7.4 directly. However, with some care, we can isolate the nonalgebraic parts of the algorithm into preprocessing and postprocessing phases, and apply the transposition principle to algebraic portions of the algorithm. We do this in the rest of the section. Before considering MODULAR POWER PROJECTION, we consider the transpose of MULTIVARIATE MULTIPOINT EVALUATION.

Theorem 7.6. *Let R be a finite ring of cardinality q given as $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ for some monic polynomial $E(Z)$. For every constant $\delta > 0$, there is an algorithm for the transpose of* MULTIVARIATE MULTIPOINT EVALUATION *with parameters d, m, N with running time $(d^m + N)^{1+\delta} \log^{1+o(1)} q$, for all d, m, N with d sufficiently large, $m \leq d^{o(1)}$, and $d^m = N$.*

Proof. We view Algorithm MULTIMODULAR-FOR-EXTENSION-RING as applying to f the linear map $\phi : R^{d^m} \to R^N$ which computes evaluations at points $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}$. Map ϕ is computed in Algorithm MULTIMODULAR-FOR-EXTENSION-RING by first performing a preprocessing phase: Step 1 produces $\tilde{f}$ with coefficients in $\mathbb{Z}[Z]$ (having degree at most $e-1$) and $\tilde{\alpha}_0, \tilde{\alpha}_1, \ldots, \tilde{\alpha}_{N-1}$ with coordinates in $\mathbb{Z}[Z]$ (having degree at most $e-1$) and then Step 2 produces $\overline{f}$ and $\overline{\alpha}_0, \overline{\alpha}_1, \ldots, \overline{\alpha}_{N-1}$, with the coefficients of $\overline{f}$ and the coordinates of each $\overline{\alpha}_i$ in $\mathbb{Z}/r'\mathbb{Z}$. Let $\tilde{\phi} : \mathbb{Z}[Z]^{d^m} \to \mathbb{Z}[Z]^N$ and $\overline{\phi} : (\mathbb{Z}/r'\mathbb{Z})^{d^m} \to (\mathbb{Z}/r'\mathbb{Z})^N$ be the multipoint evaluation maps with respect to the $\tilde{\alpha}_i$ and the $\overline{\alpha}_i$, respectively (over the appropriate domains).

The entries in the matrix associated with $\tilde{\phi}$ are products of at most $(d-1)m$ degree $(e-1)$ polynomials with nonnegative integer coefficients less than r; this matrix is applied to a vector (the coefficients of $\tilde{f}$) whose entries are degree $(e-1)$ polynomials with nonnegative integer coefficients less than r. An upper bound on the sum of d^m products of a matrix entry with a vector entry is the integer M computed in Algorithm MULTIMODULAR-FOR-EXTENSION-RING. Thus $\tilde{\phi}$ produces a vector whose entries are polynomials in $\mathbb{Z}[Z]$ with degree at most $(e-1)((d-1)m+1) \leq (e-1)dm$ and coefficients in $\{0, 1, \ldots, M-1\}$

and so does the transpose map $\tilde{\phi}^t$ (since $N = d^m$). Step 4 of MULTIMODULAR-FOR-EXTENSION-RING recovers $\tilde{\phi}(\tilde{f})$ from $\overline{\phi}(\overline{f})$, and its correctness depends only on the choice of r' as a function of the maximum degree and maximum coefficient magnitude of the entries of $\tilde{\phi}(\tilde{f})$. Thus the same procedure (which recovers elements of $\mathbb{Z}[Z]$ from elements of $\mathbb{Z}/r'\mathbb{Z}$) will recover the result of applying $\tilde{\phi}^t$ from the result of $\overline{\phi}^t$. As in Step 4, reducing each element of $\mathbb{Z}[Z]$ modulo r and $E(Z)$ gives the result in R, allowing us to finally recover the result of applying the map ϕ^t. The remainder of the present proof is thus devoted to arguing that $\overline{\phi}^t$ can be applied in the specified time via a transposed version of Algorithm MULTIMODULAR.

Algorithm MULTIMODULAR computes in t successive rounds of multimodular reduction a collection of instances of MULTIVARIATE MULTIPOINT EVALUATION. Conceptually these are organized in a tree of depth t, with the root labeled by the prime r', and the children of each nonleaf node being the primes p_i computed in Step 2 of MULTIMODULAR when its parameter "r" is the prime at the parent node. Each node v has an associated instance of MULTIVARIATE MULTIPOINT EVALUATION, comprising the polynomial $f^{(v)}$ (whose coefficients are obtained from the instance of MULTIVARIATE MULTIPOINT EVALUATION at the parent node u by lifting the coefficients of $f^{(u)}$ to the integers, and reducing modulo the prime labeling node v) and the evaluations $\alpha_0^{(v)}, \alpha_1^{(v)}, \ldots, \alpha_{N-1}^{(v)}$ (whose coordinates are similarly derived from $\alpha_0^{(u)}, \alpha_1^{(u)}, \ldots, \alpha_{N-1}^{(u)}$ by lifting to $\mathbb{Z}$ and reducing modulo the prime labeling node v). At each node v, the $\alpha_i^{(v)}$ implicitly specify the linear map $\phi^{(v)}$ that, when applied to $f^{(v)}$, gives the vector of evaluations of $f^{(v)}$ at the various $\alpha_i^{(v)}$. Each map $\phi^{(v)}$ has an associated matrix whose entries are (dm)-fold products of various $\alpha_i^{(v)}$, and an "integer lift" version in which these products are not reduced modulo the prime labeling node v.

The instances of MULTIVARIATE MULTIPOINT EVALUATION associated with the leaves of this tree are computed directly in Step 4 of Algorithm MULTIMODULAR. Specifically, for each leaf v, the map $\phi^{(v)} : \mathbb{F}_p^{d^m} \to \mathbb{F}_p^N$ (where p is the prime labeling leaf v) is applied to $f^{(v)}$ by invoking Theorem 4.1. In our algorithm for the transpose problem, we apply the transpose map $(\phi^{(v)})^t$ to $f^{(v)}$.[6] This is done within the same time bound, by Theorem 7.4, or directly by observing that the transpose of the DFT computed in Step 2 in the proof of Theorem 4.1 can again be computed rapidly using the fast multidimensional FFT for finite fields.

In the original algorithm, a postprocessing phase (successive applications of Step 5 of Algorithm MULTIMODULAR) recovers each entry in the vector $\overline{\phi}(\overline{f})$ in t successive rounds of reconstruction using the Chinese Remainder Theorem. Specifically, we work up the tree (from the leaves to the root), and at each node u we reconstruct each entry of $\phi^{(u)}(f^{(u)})$ from the corresponding entries of $\phi^{(v)}(f^{(v)})$ as v ranges over the children of u. Correctness at each such step comes from the fact that the product of the primes labeling the children of u exceeds $d^m(r-1)^{md}$, which is an upper bound on the sum of d^m products, each being the product of an entry of the integer lift matrix associated with $\phi^{(u)}$ with the integer lift of an entry from the vector $f^{(u)}$. In our algorithm for the transpose problem, we perform the same reconstruction, working up the tree, and reconstructing each entry of $(\phi^{(u)})^t(f^{(u)})$ from the corresponding entries of $(\phi^{(v)})^t(f^{(v)})$ as v ranges over the children of u. For correctness we need that the product of the primes labeling the children of u exceeds an upper bound on the sum of N products, each being the product of the *transposed* integer lift matrix associated with $\phi^{(u)}$ with the integer lift of an entry from the vector $f^{(u)}$. Since $d^m = N$, $d^m(r-1)^{md}$ is again a valid upper bound, and we conclude that each Chinese Remainder Theorem reconstruction step succeeds, eventually yielding (at the root) $\overline{\phi}^t(\overline{f})$ as desired.

Because our overall algorithm for the transpose of MULTIVARIATE MULTIPOINT EVALUATION entails the same computations in the pre- and post- processing phases as the nontransposed version, and the com-

[6]Formally, we should have a different name for the vector to which the transpose map is applied, which has N entries instead of d^m. But because $N = d^m$, and to avoid clutter in the proof, we will continue to use f.

putation of the transposed MULTIVARIATE MULTIPOINT EVALUATION instances at the leaves can be performed in the same time bound as the nontransposed version, we obtain an algorithm for the transpose of MULTIVARIATE MULTIPOINT EVALUATION, when $N = d^m$, with the same running time bound as stated in Corollary 4.5. $\square$

Theorem 7.7. *Let R be a finite ring of cardinality q given as $(\mathbb{Z}/r\mathbb{Z})[Z]/(E(Z))$ for some monic polynomial $E(Z)$. For every constant $\delta > 0$, if we have access to $d^{1+\delta}$ distinct elements of R whose differences are units in R, there is an algorithm for* MODULAR POWER PROJECTION *over R with running time $d^{1+\delta}\log^{1+o(1)} q$, for sufficiently large d.*

Proof. Consider first the reduction from MODULAR COMPOSITION (with parameters $d, m = 1, N = d$) to MULTIVARIATE MULTIPOINT EVALUATION of Theorem 3.1. An instance of MODULAR COMPOSITION is specified by degree d polynomials $f(X), g(X), h(X)$. We describe the reduction as the product of linear maps applied to the vector of coefficients of f. Steps 2 and 3 do not involve f, and can be executed in a preprocessing phase.

Step 1 is given by $\phi_1 : R^d \to R^{d'}$ which maps f to f' by permuting the coefficients and padding with 0's (here $d' = d_0^\ell \geq d$). Step 4 is given by $\phi_4 : R^{d'} \to R^{N'}$ which maps f' to its evaluations at the $N' > d'$ evaluation points (the α's). Step 5 is given by $\phi_5 : R^{N'} \to R^{N'}$ which maps these evaluations to the coefficients of the unique univariate polynomial having these values at the β's. Step 6 is given by $\phi_6 : R^{N'} \to R^d$ which maps the resulting degree $N' - 1$ univariate polynomial to its reduction modulo $h(X)$. All of $\phi_1, \phi_4, \phi_5, \phi_6$ are linear maps, and thus the overall algorithm for MODULAR COMPOSITION (after the preprocessing phase involving $g(X)$ and $h(X)$) can be described as the linear map $\phi_6 \circ \phi_5 \circ \phi_4 \circ \phi_1 : R^d \to R^d$.

We are interested in computing the transposed map $\phi_1^t \circ \phi_4^t \circ \phi_5^t \circ \phi_6^t : R^d \to R^d$. We argue that transposed map can be computed in time comparable to the time required for the nontransposed map. In Theorem 3.1, ϕ_6 is computed rapidly using fast polynomial division with remainder. By the transposition principle (Theorem 7.4), ϕ_6^t can be computed in comparable time. In Theorem 3.1, ϕ_5 is computed rapidly using fast univariate polynomial interpolation. By the transposition principle (Theorem 7.4), ϕ_5^t can be computed in comparable time.

In Theorem 3.1, ϕ_4 is computed rapidly by invoking a fast algorithm for MULTIVARIATE MULTIPOINT EVALUATION. We claim that ϕ_4^t can be computed in the time expended by Algorithm MULTIMODULAR-FOR-EXTENSION-RING to compute ϕ_4. We'd like to apply Theorem 7.6, but that requires a "square" instance (i.e., one in which $d^m = N$, which gives rise to a linear map represented by a square matrix), but in our case N' is larger than d'. But, just as we could have computed ϕ_4 by invoking Algorithm MULTIMODULAR-FOR-EXTENSION-RING N'/d' times with d' evaluation points each time, we can compute ϕ_4^t by computing the transpose of a N'/d' square instances (via Theorem 7.6) and summing the resulting vectors.

Finally, ϕ_1^t is just a projection followed by a permutation of the coordinates, which can trivially be computed in time comparable to that required for computing ϕ_1. $\square$

Remark. There are explicit algorithms known for ϕ_5^t (transposed univariate interpolation) and ϕ_6^t (transposed univariate polynomial division with remainder) (see, e.g., [BLS03]), and our algorithm in Theorem 7.6 is also explicit. Thus we have an explicit algorithm for MODULAR POWER PROJECTION (whereas in general, use of the transposition principle may produce an algorithm that can only be written down by manipulating the linear straight-line program).

8 Applications

In this section, we describe some improved algorithms that arise as a consequence of our new algorithms for MODULAR COMPOSITION and MODULAR POWER PROJECTION. To emphasize the fact that modular composition and modular power projection occur as black boxes within these algorithms, we write $C(n,q)$ and $P(n,q)$ for the number of bit operations required to perform a modular composition and a modular power projection, respectively, of degree n polynomials over $\mathbb{F}_q$. As shown by Corollary 7.2 (and using the remark following it), we now have $C(n,q) \le n^{1+o(1)} \log^{1+o(1)} q$. Similarly, by Theorem 7.7 we have $P(n,q) \le n^{1+o(1)} \log^{1+o(1)} q$.

Note that all of the algorithms we describe in this section are algebraic except for the steps that use our multimodular reduction-based algorithm for MODULAR COMPOSITION or MODULAR POWER PROJECTION. Consequently, in characteristic $p \le n^{o(1)}$, we may instead use the second part of Corollary 7.2 and Theorem 7.5 to produce completely algebraic algorithms; to obtain an upper bound on algebraic operation counts for these, remove a factor of $\log q$ from the bit operation counts we state in this section.

8.1 Polynomial factorization

We start with the flagship application, to the problem of polynomial factorization.

There are three stages in variants of the Cantor-Zassenhaus algorithm for factoring a degree n univariate polynomial over $\mathbb{F}_q$: squarefree factorization, distinct-degree factorization, and equal-degree factorization. The first stage, squarefree factorization, can be performed in $n^{1+o(1)} \log^{2+o(1)} q$ bit operations, using an algorithm attributed by [KS98] to Yun. The second stage, distinct-degree factorization, has a deterministic algorithm due to Kaltofen & Shoup [KS98] that takes

$$n^{0.5+o(1)} C(n,q) + M(n) \log^{2+o(1)} q$$

bit operations, as described below. The third stage, equal-degree factorization, has a randomized algorithm due to von zur Gathen & Shoup [vzGS92] that takes an expected number of $n^{1+o(1)} \log^{2+o(1)} q + C(n,q) \log n$ bit operations.

Notice that with our bound $C(n,q) = n^{1+o(1)} \log^{1+o(1)} q$, the first and third stages use $n^{1+o(1)} \log^{2+o(1)} q$ bit operations and the second stage improves to

$$n^{1.5+o(1)} \log^{1+o(1)} q + n^{1+o(1)} \log^{2+o(1)} q$$

bit operations. The second stage remains the barrier to an "exponent 1" algorithm, so we describe the algorithm of Kaltofen & Shoup in enough detail here (and in a manner differing somewhat from the original) to highlight a self-contained open problem whose resolution would improve its efficiency to $n^{1+o(1)} \log^{2+o(1)} q$ bit operations. This will also illustrate the critical role played by MODULAR COMPOSITION in this algorithm.

The problem we are trying to solve is:

Problem 8.1 (DISTINCT-DEGREE FACTORIZATION). *Given a monic, squarefree polynomial $f \in \mathbb{F}_q[X]$ of degree n, output $f_1, f_2, \ldots, f_n \in \mathbb{F}_q[X]$ where f_i is either 1 or the product of degree-i irreducible polynomials, and $f_1 f_2 \cdots f_n = f$.*

The crucial (standard) algebraic fact used in these algorithms is:

Proposition 8.2. *The polynomial $s_i(X) \overset{\text{def}}{=} (X^{q^i} - X) \in \mathbb{F}_q[X]$ is the product of all monic irreducible polynomials over $\mathbb{F}_q$ whose degree divides i.*

Therefore, computing $\gcd(s_i(X), f(X))$ splits off those irreducible factors of f whose degrees divide i. In preparing the polynomial $s_i(X)$ for this purpose, we are free to compute it modulo $f(X)$.

The main step in the algorithm for DISTINCT-DEGREE FACTORIZATION will be to split the input polynomial f into two nonconstant polynomials $f_1 f_2 \cdots f_m$ and $f_{m+1} f_{m+2} \cdots f_n$ for some $m \in \{1, 2, \ldots, n\}$. One could do this by computing $\gcd(s_i(X), f(X))$ for $i = 1, 2, \ldots, n$ and stopping at the first nontrivial gcd, but in the worst case, a nontrivial split will not be found until $i \approx n/2$ which spoils any chance of a subquadratic algorithm. Instead, we will perform a "binary search": we begin with $m = n/2$, and if this does not yield a nontrivial split, we proceed to either $m = n/4$ or $m = 3n/4$ depending on whether $f_1 f_2 \cdots f_{n/2}$ equals f or 1, and so on.

For this purpose we need to be able to solve the following subproblem, which gives us the polynomials needed to compute the "splits" on-the-fly in the above binary-search strategy (and note that for our intended application we do not care if the $s_i(X)$ factors are repeated, which explains the a_i's below):

Problem 8.3. *Given a monic, squarefree polynomial $f \in \mathbb{F}_q[X]$ of degree n, a positive integer m, and the polynomial $X^q \bmod f(X)$, compute a polynomial*

$$s_1(X)^{a_1} \cdot s_2(X)^{a_2} \cdot \cdots \cdot s_m(X)^{a_m} \bmod f(X) = \prod_{i=1}^{m} (X^{q^i} - X)^{a_i} \bmod f(X)$$

for some positive integers a_i.

It is easy to see that this problem can be solved in

$$m \cdot O\left(C(n, q) + M(n) \log^{1+o(1)} q\right)$$

bit operations: with m successive modular compositions with X^q, we can obtain $X^{q^i} \bmod f(X)$ for $i = 1, 2, \ldots, m$, and then m further polynomial additions and multiplications modulo f suffice to compute $\prod_{i=1}^{m}(X^{q^i} - X) \bmod f(X)$.

Kaltofen & Shoup describe a clever algorithm that reduces the exponent on m from 1 to $1/2$:

Lemma 8.4 (implicit in [KS98])**.** *Problem 8.3 can be solved in*

$$O\left(C(n, q)\sqrt{m} + M(n) M(\sqrt{m}) \log \sqrt{m} \log^{1+o(1)} q\right)$$

operations.

Proof. Refer to Figure 1 for the running times of the operations on polynomials we use in this proof.

Put $k = m - \lfloor \sqrt{m} \rfloor^2 \leqslant 2\sqrt{m}$. First, compute X^{q^i} modulo $f(X)$ for $i = 0, 1, 2, \ldots, \lfloor \sqrt{m} \rfloor - 1$; then compute $X^{q^{j\lfloor \sqrt{m} \rfloor}}$ modulo $f(X)$ for $j = 1, 2, \ldots, \lfloor \sqrt{m} \rfloor$. This requires $O(C(n, q)\sqrt{m})$ bit operations, since we are given $X^q \bmod f(X)$ to begin with. At this point we have computed $X^{q^{m-k}} \bmod f(X)$; now compute $X^{q^{m-i}} \bmod f(X)$ for $i = k - 1, \ldots, 0$, using $O(k) = O(\sqrt{m})$ further compositions with $X^q \bmod f(X)$, at a cost of $O(C(n, q)\sqrt{m})$ bit operations. Form the product

$$Q_0(X) = \prod_{i=0}^{k-1} (X^{q^{m-i}} - X) \bmod f(X)$$

using $O(M(n)\sqrt{m} \log^{1+o(1)} q)$ bit operations.

Form the degree $\lfloor\sqrt{m}\rfloor$ polynomial $P(Z)$ over the ring $\mathbb{F}_q[X]/(f(X))$ defined as:

$$P(Z) \overset{\text{def}}{=} \prod_{i=0}^{\lfloor\sqrt{m}\rfloor-1} (Z - X^{q^i}) \bmod f(X).$$

This requires $O(M(\sqrt{m})\log\sqrt{m})$ operations in the ring, or $O\left(M(n)M(\sqrt{m})\log\sqrt{m}\log^{1+o(1)} q\right)$ bit operations. Evaluate $P(Z)$ at the elements $X^{q^{j\lfloor\sqrt{m}\rfloor}} \bmod f(X)$ for $j = 1, 2, \ldots, \lfloor\sqrt{m}\rfloor$, and take the product of these evaluations modulo $f(X)$, yielding:

$$Q_1(X) = \prod_{j=1}^{\lfloor\sqrt{m}\rfloor}\prod_{i=0}^{\lfloor\sqrt{m}\rfloor-1} (X^{q^{j\lfloor\sqrt{m}\rfloor}} - X^{q^i}) \bmod f(X)$$

which equals:

$$\prod_{j=1}^{\lfloor\sqrt{m}\rfloor}\prod_{i=0}^{\lfloor\sqrt{m}\rfloor-1} (X^{q^{j\lfloor\sqrt{m}\rfloor-i}} - X)^{q^i} \bmod f(X).$$

Using fast multipoint evaluation, this step entails $O(M(\sqrt{m})\log\sqrt{m})$ operations in the ring, or

$$O\left(M(n)M(\sqrt{m})\log\sqrt{m}\log^{1+o(1)} q\right)$$

bit operations. Finally, multiply $Q_0(X)$ with $Q_1(X)$ to obtain a polynomial of the desired form (the a_i are various powers of q). $\square$

Using Problem 8.3 as a subroutine, it is not hard to describe a fast algorithm for DISTINCT-DEGREE FACTORIZATION:

Theorem 8.5. *If Problem 8.3 can be solved in $O\left(n^\alpha m^\beta \log^{1+o(1)} q\right)$ bit operations (with $\alpha > 1$), then there is an algorithm for* DISTINCT-DEGREE FACTORIZATION *that uses*

$$O\left(n^{\alpha+\beta}\log^4 n + M(n)\log q\right)\cdot\log^{1+o(1)} q$$

bit operations.

Proof. We first prepare the polynomial $X^q \bmod f(X)$ needed as input to Problem 8.3, by repeated squaring, at a cost of $O(M(n)\log q)\cdot\log^{1+o(1)} q$ bit operations.

Now, in addition to the input of a squarefree $f(X) \in \mathbb{F}_q[X]$ of degree n, we assume we are given a range within which we know all of the degrees of the irreducible factors of f lie. Initially, this is just $1\ldots n$.

If the range consists of only a single integer, then we can output $f(X)$ itself and halt. Otherwise, set m to the midpoint of this range, and compute a polynomial as specified in Problem 8.3; call this polynomial $S(X)$. Compute $\gcd(S(X), f(X))$. If this gcd is $f(X)$, then we reduce the range to the first half and recurse; if this gcd is a constant polynomial, then we reduce the range to the second half and recurse; if this gcd is a nontrivial polynomial $f_{\text{lower}}(X)$, then we compute $f_{\text{upper}}(X) = f(X)/f_{\text{lower}}(X)$, and these two polynomials represent a successful "split." Notice that $\deg(f_{\text{lower}}) + \deg(f_{\text{upper}}) = \deg(f)$. We now recurse on f_{lower} (with the range reduced to the first half) and f_{upper} (with the range reduced to the second half).

We now analyze the operation count of this recursive algorithm when factoring a degree n input polynomial. Notice that we never set m larger than n throughout the entire algorithm, so we will pessimistically assume it is always n to simplify the analysis.

Let $T(n', r)$ denote the bit operations used by the procedure, when called with a polynomial of degree n' and range of size r. If $r = 1$, the cost is zero. Otherwise, the procedure solves Problem 8.3 at a cost of at most $c_1 n'^{\alpha} n^{\beta} \log^{1+o(1)} q$, and the other operations before the recursive call (a gcd, and possibly a polynomial division) cost at most $c_2 n' \log^3 n' \log^{1+o(1)} q$ for some constants c_1, c_2. Set $c = c_1 + c_2$.

We will prove that for all $T(n', r)$ with $n', r \leq n$,

$$T(n', r) \leq c n'^{\alpha} (\log^3 n') n^{\beta} (\log r) \log^{1+o(1)} q,$$

by induction on r. The base case, when $r = 1$, is clear. In general we have that

$$T(n', r) \leq \left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n'\right) \log^{1+o(1)} q + \max_{1<i<n'} \begin{cases} T(n', r/2) \\ T(i, r/2) + T(n' - i, r/2) \end{cases}$$

where the two lines in the inequality correspond to the cases that result in recursive calls. In the first case we have:

$$\begin{aligned} &\left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n'\right) \log^{1+o(1)} q + T(n', r/2) \\ &\qquad \leq \left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n' + c n'^{\alpha} \log^3 n' n^{\beta} (\log r - 1)\right) \log^{1+o(1)} q \\ &\qquad \leq c n'^{\alpha} (\log^3 n') n^{\beta} (\log r) \log^{1+o(1)} q \end{aligned}$$

as required. In the second case, we have:

$$\begin{aligned} &\left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n'\right) \log^{1+o(1)} q + T(i, r/2) + T(n' - i, r/2) \\ &\quad \leq \left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n' + c[i^{\alpha} \log^3 i + (n' - i)^{\alpha} \log^3 (n' - i)] n^{\beta} (\log r - 1)\right) \log^{1+o(1)} q \\ &\quad \leq \left(c_1 n'^{\alpha} n^{\beta} + c_2 n' \log^3 n' + c[n'^{\alpha} \log^3 n'] n^{\beta} (\log r - 1)\right) \log^{1+o(1)} q \\ &\quad \leq c n'^{\alpha} (\log^3 n') n^{\beta} (\log r) \log^{1+o(1)} q \end{aligned}$$

as required. The claimed upper bound in the theorem follows by considering $T(n, n)$. ☐

We now see how our new modular composition algorithm yields the fastest univariate factorization algorithm that works over arbitrary finite fields:

Theorem 8.6. *There is a randomized algorithm that returns the irreducible factors of a degree n polynomial $f \in \mathbb{F}_q[X]$ and uses an expected*

$$\left(n^{1.5+o(1)} + n^{1+o(1)} \log q\right) \cdot \log^{1+o(1)} q$$

bit operations.

Proof. As noted above, the first and third phases already fall within this bound. Plugging Corollary 7.2 into Lemma 8.4 yields an algorithm for Problem 8.3 using $n^{1+o(1)} m^{0.5+o(1)} \log^{1+o(1)} q$ bit operations. Theorem 8.5 then yields the claimed result. ☐

We consider it a very interesting open problem to devise an algorithm for Problem 8.3 that takes only $n^{1+o(1)} m^{o(1)} \log^{1+o(1)} q$ bit operations. By Theorem 8.5, this would give a randomized algorithm for factoring a degree n polynomial over $\mathbb{F}_q$ requiring an expected $n^{1+o(1)} \log^{2+o(1)} q$ bit operations.

8.2 Irreducibility testing

In this problem we are given $f(X) \in \mathbb{F}_q[X]$ of degree n, and we want to determine whether or not it is irreducible. Rabin's algorithm [Rab80] can be implemented (deterministically) to take

$$n^{1+o(1)} \log^{2+o(1)} q + C(n,q) \cdot O(\log^2 n)$$

bit operations [vzGG99], so we obtain a running time of $n^{1+o(1)} \log^{2+o(1)} q$. This becomes the fastest known (up to lower order terms) for all n and q, and it constitutes an asymptotic improvement over the running time of previous implementations when $\log q < n^{0.688}$.

8.3 Manipulation of normal bases

A *normal element* in the extension field $\mathbb{F}_q[X]/(h(X))$, where h is monic and irreducible of degree n, is an element α for which $\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}$ form a basis for the extension field over $\mathbb{F}_q$; such a basis is called a *normal basis*.

Kaltofen & Shoup [KS98] study three natural problems related to manipulating normal bases: the problem of *basis selection* (given a degree n irreducible $h(X)$, find a normal element of $\mathbb{F}_q[X]/(h(X))$); and the problems of converting to a normal-basis representation from a power-basis representation, and vice versa. The algorithms in [KS98] rely on two problems defined in that paper:

- Automorphism evaluation: given degree $n-1$ polynomials $f(X), g(X)$ and degree n polynomial $h(X)$, all in $\mathbb{F}_q[X]$, output the degree $n-1$ polynomial $\sum_{i=0}^{n-1} f_i(g(X)^{q^i} \bmod h(X))$, where the f_i are the coefficients of $f(X)$ (i.e., $f(X) = \sum_{i=0}^{n-1} f_i X^i$).
- Automorphism projection: given a linear form $\tau : \mathbb{F}_q^n \to \mathbb{F}_q$, a degree $n-1$ polynomial $g(X)$ and a degree n polynomial $h(X)$, both in $\mathbb{F}_q[X]$, output $(\tau(g(X)^{q^i} \bmod h(X))$ for $i = 0, 1, \ldots, n-1$. Here we identify a polynomial with the vector of its coefficients.

The two problems are the transpose of each other, and bear a resemblance to MODULAR COMPOSITION and MODULAR POWER PROJECTION, respectively (here the $g(X)$ polynomial is raised to successive q-th powers, rather than consecutive powers). Kaltofen & Shoup [KS98] describe explicit baby-steps/giant-steps algorithms for the two algorithms that rely on fast matrix multiplication (a la Brent & Kung) *and* MODULAR COMPOSITION and MODULAR POWER PROJECTION. In particular, their algorithms yield running times of

$$O\left(C(n,q)n^{1/2} + (n^{\omega_2/2} + M(n)\log q)\log^{1+o(1)} q\right)$$

for automorphism evaluation, and

$$O\left(C(n,q)n^{1/2} + P(n,q)n^{1/2} + (n^{\omega_2/2} + M(n)\log q)\log^{1+o(1)} q\right)$$

for automorphism projection (recall the definition of ω_2 from Section 1.1). With our algorithms for MODULAR COMPOSITION and MODULAR POWER PROJECTION (and noting that $\omega_2 \geq \omega + 1 > 3$), both problems can be solved in time

$$n^{\omega_2/2} \log^{1+o(1)} q + n^{1+o(1)} \log^{2+o(1)} q. \tag{8.1}$$

The algorithms of [KS98] for manipulating normal bases have running times that are dominated by the invocations of automorphism evaluation and projection. Thus the three problems — of finding a normal element, converting from power-basis coordinates to normal-basis coordinates, and converting from normal-basis coordinates to power-basis coordinates — have running times bounded by (8.1) (the first and second are randomized, with this expression bounding the expected running time). These running times represent improvements over [KS98] and are the current fastest algorithms for these problems, when $\log q < n$.

Remark. In the algorithms in the previous three subsections, the quadratic dependence on $\log q$ (which is nonoptimal) arises solely from the need to compute X^q modulo some degree n polynomial $f \in \mathbb{F}_q[X]$ (specifically, the polynomial to be factored or the polynomial being tested for irreducibility). This is done by repeated squaring at a cost of $O(M(n)\log q)$ $\mathbb{F}_q$-operations. An insight of Kaltofen & Shoup [KS97] is that when $q = p^k$, and assuming that $\mathbb{F}_q$ is represented explicitly as $\mathbb{F}_p[Z]/(E(Z))$ for some degree k irreducible $E \in \mathbb{F}_p[Z]$, this term can be improved as follows.

We illustrate the idea when k is a power of 2. Define $g_i(X) \stackrel{\text{def}}{=} X^{p^{2^i}} \bmod f(X)$, and let $\sigma : \mathbb{F}_q \to \mathbb{F}_q$ denote the Frobenius map $x \mapsto x^p$. As in Section 6, denote by $\sigma^j(g_i)$ the polynomial g_i with σ^j applied to each of its coefficients. Define $h_i(Z) \stackrel{\text{def}}{=} Z^{p^{2^i}} \bmod E(Z)$ (so h_i is the polynomial representation of the map σ^{2^i}). We have that

$$g_i(X) = \sigma^{2^{i-1}}(g_{i-1})(g_{i-1}(X)) \bmod f(X),$$

and note that $g_{\log k}(X)$ is the desired polynomial $X^q \bmod f(X)$.

We can compute $g_{\log k}$ as follows. First, compute $g_0(X) = X^p \bmod f(X)$ and $h_0(Z) = Z^p \bmod E(Z)$ using repeated squaring. Then, for $i = 1, 2, \ldots, \log k$, compute $g_i(X) = \sigma^{2^{i-1}}(g_{i-1})(g_{i-1}(X)) \bmod f(X)$, and $h_i(Z) = h_{i-1}(h_{i-1}(Z)) \bmod E(Z)$. The latter computation entails a single modular composition of degree k polynomials over $\mathbb{F}_p$; each coefficient of the polynomial $\sigma^{2^{i-1}}(g_{i-1})$ can be obtained from g_{i-1} by a modular composition of the degree k polynomial representing the coefficient with h_{i-1}, and then g_i is obtained with a single modular composition of degree n polynomials over $\mathbb{F}_q$.

The overall cost is

$$O\left(\log p\left(M(n)\log^{1+o(1)} q + M(k)\log^{1+o(1)} p\right)\right)$$

bit operations to compute g_0 and h_0, plus $(n+1)C(k,p) + C(n,q)$ bit operations for each of the $\log k$ iterations, for a total of

$$k^{1+o(1)} n^{1+o(1)} \log^{2+o(1)} p$$

bit operations, using our new algorithms for MODULAR COMPOSITION. This should be contrasted with the $k^{2+o(1)} n^{1+o(1)} \log^{2+o(1)} p$ bit operations for the standard repeated squaring approach. Thus, in fixed characteristic, the nonoptimal quadratic dependence on $\log q$ of the algorithms in the previous three subsections can be replaced with the optimal one (up to lower order terms), using this idea.

8.4 Computing minimal polynomials

In this problem, we are given $g(X), h(X) \in \mathbb{F}_q[X]$, both of degree at most n, and we must output the minimal polynomial of $g(X)$ in the ring $\mathbb{F}_q[X]/(h(X))$; i.e., the monic polynomial $f(X)$ of minimal degree for which $f(g(X)) \bmod h(X) = 0$. Shoup's randomized algorithm [Sho99] can be implemented to run in expected time

$$O(M(n)\log n \log^{1+o(1)} q + C(n,q) + P(n,q)),$$

so we obtain an expected running time of $n^{1+o(1)} \log^{1+o(1)} q$ using our algorithms for MODULAR COMPOSITION and MODULAR POWER PROJECTION, which is best possible up to lower order terms.

8.5 Generating irreducible polynomials

In this problem, we are given a finite field $\mathbb{F}_q$ and a positive integer d, and we must output an irreducible polynomial of degree d over $\mathbb{F}_q$. Using our algorithm for MODULAR COMPOSITION, Couveignes and Lercier [CL] have very recently given a randomized algorithm for this problem with expected running time

$d^{1+o(1)} \log^{5+o(1)} q$. While the dependence on $\log q$ is not best possible up to lower order terms, this algorithm is the first to achieve nearly-linear complexity in the degree d.

8.6 Frobenius evaluation

The fact that our algorithm applies to extension rings, not just to finite fields, leads to some additional applications. One example, suggested to us by Hendrik Hubrechts, is that of *Frobenius evaluation*. (See [Hub] for some related applications in p-adic arithmetic.)

Let $P(X) \in (\mathbb{Z}/p^n\mathbb{Z})[X]$ be a monic polynomial whose reduction modulo p is irreducible. Then the ring $R = (\mathbb{Z}/p^n\mathbb{Z})[X]/(P(X))$ admits a unique *Frobenius endomorphism* $F : R \to R$ satisfying $F(r) \equiv r^p \pmod{p}$ for all $r \in R$. Once one has computed the image of $X \in R$ under F, one can then evaluate F efficiently on any element of R by using modular composition.

In more number-theoretic language, the ring R arises as the quotient modulo p^n of an unramified extension of the ring $\mathbb{Z}_p$ of p-adic integers. (The existence of the Frobenius endomorphism is a consequence of Hensel's lemma.) Consequently, an algorithm for evaluating F efficiently leads to improvements in certain algorithms based on p-adic analysis. An explicit example occurs in Hubrechts's computation of zeta functions of hyperelliptic curves over finite fields, using deformations in p-adic Dwork cohomology: substituting for our modular composition algorithm in [Hub08, §6.2] leads to a runtime improvement therein.

9 Open problems

We conclude with some open problems.

- Our algorithm for MULTIVARIATE MULTIPOINT EVALUATION is only optimal up to lower order terms in case $m \leqslant d^{o(1)}$. It would be interesting to describe a near-optimal algorithm in the remaining cases, or perhaps just the multilinear case to start. It would also be satisfying to give a near-optimal *algebraic* algorithm for MULTIVARIATE MULTIPOINT EVALUATION in arbitrary characteristic, not just small characteristic.

- It would also be interesting to adapt our algebraic algorithms so that they work in a commutative *ring* of small characteristic. Currently we require a field (see the discussion following Eq. (6.2)).

- As noted earlier, the reduction from MODULAR COMPOSITION to MULTIVARIATE MULTIPOINT EVALUATION plays an important role in our work because it is easier to control the growth of integers when solving the lifted version of MULTIVARIATE MULTIPOINT EVALUATION. One wonders whether there are other problems involving polynomials that can exploit the combination of transforming the problem to a multivariate version with smaller total degree, and then lifting to characteristic zero followed by multimodular reduction. For instance, can such techniques be profitably applied to other problems whose currently best algorithms use a "baby steps/giant steps" technique in the manner of [BK78]? We have specifically in mind such problems as *automorphism projection* and *automorphism evaluation* as defined in [KS98], and discussed in Section 8.3.

- As noted earlier, an algorithm for Problem 8.3 using only $n^{1+o(1)}m^{o(1)} \log^{1+o(1)} q$ bit operations would lead to a randomized algorithm for factoring a degree n polynomial over $\mathbb{F}_q$ using $n^{1+o(1)} \log^{2+o(1)} q$ expected bit operations. It seems that giving an algorithm for Problem 8.3 with operation count $n^{1+o(1)}m^{\beta}$ for any constant $\beta < 1/2$, even under an assumption of small characteristic, will require a new idea. Another route to an "exponent 1" algorithm for polynomial factorization would be to give "exponent 1" algorithms for automorphism projection and automorphism evaluation, and then use the implementation described in [KS98] of the so-called Black Box Berlekamp algorithm for polynomial factorization.

10 Acknowledgements

We thank Henry Cohn, Joachim von zur Gathen, David Harvey, Erich Kaltofen, and Eyal Rozenman for useful discussions, and Éric Schost for helpful comments on a draft of [Uma08]. We thank Swastik Kopparty and Madhu Sudan for some references mentioned in Section 5, and Ronald de Wolf and the FOCS 2008 referees for helpful comments on the conference paper [KU08]. We thank Ariel Gabizon, Hendrik Hubrechts, Dieter Theunckens, and the anonymous referees for helpful comments, and Igor Sergeev for identifying an error in Section 6. Finally, we thank Madhu Sudan for hosting a visit of the second author to MIT, which launched this collaboration.

References

[BCS97] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1997.

[Bel61] E. G. Belaga. Evaluation of polynomials of one variable with preliminary preprocessing of the coefficients. *Problemy Kibernet.*, 5:7–15, 1961.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–735, 1970.

[Ber98] D. J. Bernstein. Composing power series over a finite ring in essentially linear time. *J. Symb. Comput.*, 26(3):339–341, 1998.

[Ber08] D. J. Bernstein. Fast multiplication and its applications. In Joseph P. Buhler and Peter Stevenhagen, editors, *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography*, pages 325–384. Cambridge Univ. Press, 2008.

[BK78] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25(4):581–595, 1978.

[BLS03] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC '03: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 37–44, New York, NY, USA, 2003. ACM.

[CL] J.-M. Couveignes and R. Lercier. Fast construction of irreducible polynomials over finite fields. Available at `http://arxiv.org/abs/0905.1642`.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[CZ81] D.G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[GM07] A. Gál and P. B. Miltersen. The cell probe complexity of succinct data structures. *Theor. Comput. Sci.*, 379(3):405–417, 2007.

[vzG06] J. von zur Gathen. Who was who in polynomial factorization. In Barry M. Trager, editor, *ISSAC*, page 2. ACM, 2006.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[vzGP01] J. von zur Gathen and D. Panario. Factoring polynomials over finite fields: A survey. *J. Symb. Comput.*, 31(1/2):3–17, 2001.

[vzGS92] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Computational Complexity*, 2:187–224, 1992.

[GR06] V. Guruswami and A. Rudra. Explicit capacity-achieving list-decodable codes. In Jon M. Kleinberg, editor, *STOC*, pages 1–10. ACM, 2006.

[HP98] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[Hub] H. Hubrechts. Fast arithmetic in unramified p-adic fields. in preparation.

[Hub08] H. Hubrechts. Point counting in families of hyperelliptic curves. *Foundations of Computational Math.*, 8(1):137–169, 2008.

[Kal03] E. Kaltofen. Polynomial factorization: a success story. In J. Rafael Sendra, editor, *ISSAC*, pages 3–4. ACM, 2003.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KS97] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *ISSAC*, pages 184–188, 1997.

[KS98] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of Computation*, 67(223):1179–1197, 1998.

[KU08] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *FOCS*, pages 481–490. IEEE Computer Society, 2008.

[Mil95] P. B. Miltersen. On the cell probe complexity of polynomial evaluation. *Theor. Comput. Sci.*, 143(1):167–174, 1995.

[NZ04] M. Nüsken and M. Ziegler. Fast multipoint evaluation of bivariate polynomials. In Susanne Albers and Tomasz Radzik, editors, *ESA*, volume 3221 of *Lecture Notes in Computer Science*, pages 544–555. Springer, 2004.

[Pan66] V. Ya. Pan. Methods of computing values of polynomials. *Russian Math. Surveys*, 21(1):105–136, 1966.

[PT09] D. Panario and D. Thomson. Efficient pth root computations in finite fields of characteristic p. *Des. Codes Cryptogr.*, 50:351–358, 2009.

[PV05] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *FOCS*, pages 285–294. IEEE Computer Society, 2005.

[Rab80] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput.*, 9(2):273–280, 1980.

[Sho] V. Shoup. NTL 5.5. Available at `http://www.shoup.net/ntl/`.

[Sho94] V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symb. Comput.*, 17(5):371–391, 1994.

[Sho99] V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *ISSAC*, pages 53–58, 1999.

[Sho08] V. Shoup. *A Computational Introduction to Number Theory and Algebra (version 2.3)*. Cambridge University Press, 2008. Available at `http://www.shoup.net/ntb/`.

[Uma08] C. Umans. Fast polynomial factorization and modular composition in small characteristic. In Richard E. Ladner and Cynthia Dwork, editors, *STOC*, pages 481–490. ACM, 2008.