MATHEMATICS OF COMPUTATION
Volume 00, Number 0, Pages 000–000
S 0025-5718(XX)0000-0

COMPUTING MODULAR POLYNOMIALS IN QUASI-LINEAR TIME

ANDREAS ENGE

Abstract. We analyse and compare the complexity of several algorithms for computing modular polynomials. Under the assumption that rounding errors do not influence the correctness of the result, which appears to be satisfied in practice, we show that an algorithm relying on floating point evaluation of modular functions and on interpolation has a complexity that is up to logarithmic factors linear in the size of the computed polynomials. In particular, it obtains the classical modular polynomial Φ_ℓ of prime level ℓ in time

$$O\left(\ell^2 \log^3 \ell \, M(\ell)\right) \subseteq O\left(\ell^3 \log^{4+\varepsilon} \ell\right),$$

where $M(\ell)$ is the time needed to multiply two ℓ-bit numbers.

Besides treating modular polynomials for $\Gamma^0(\ell)$, which are an important ingredient in many algorithms dealing with isogenies of elliptic curves, the algorithm is easily adapted to more general situations. Composite levels are handled just as easily as prime levels, as well as polynomials between a modular function and its transform of prime level, such as the Schläfli polynomials and their generalisations.

Our distributed implementation of the algorithm confirms the theoretical analysis by computing modular equations of record level around 10000 in less than two weeks on ten processors.

1. Definitions and main result

Modular polynomials, in their broadest sense, are bivariate polynomials with a pair of modular functions as zero. Given any two modular functions f and g for arbitrary congruence subgroups, the function fields $\mathbb{C}(f)$ and $\mathbb{C}(g)$ are finite extensions of $\mathbb{C}(j)$, so that there are two polynomials relating f resp. g to j. Taking the resultant of these polynomials with respect to j, one sees that a polynomial relationship between f and g exists. In practice, one is rather interested in the minimal polynomial of f over $\mathbb{C}(g)$, say, that will be called the modular polynomial of f with respect to g. If the functions satisfy some conditions on the rationality and integrality of their q-expansion coefficients, the modular polynomial has rational integral coefficients.

Different modular polynomials parameterise moduli spaces related to elliptic curves. Let $\Gamma = \mathrm{Sl}_2(\mathbb{Z})/\{\pm 1\} = \mathrm{PSl}_2(\mathbb{Z})$ be the full modular group, and $\mathbb{C}_\Gamma = \mathbb{C}(j)$ the field of modular functions invariant under Γ; j itself parameterises isomorphism classes of elliptic curves. Of special interest for applications is the congruence

Received by the editor April 24, 2007; revised May 5, 2008.
2000 *Mathematics Subject Classification.* Primary 11Y16, secondary 11G15.

subgroup

$$\Gamma^0(\ell) = \begin{pmatrix} 1 & 0 \\ 0 & \ell \end{pmatrix}^{-1} \Gamma \begin{pmatrix} 1 & 0 \\ 0 & \ell \end{pmatrix} \cap \Gamma = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma : \ell | b \right\}$$

for ℓ prime; its modular polynomial parameterises isomorphism classes of elliptic curves together with an isogeny of degree ℓ. This is heavily used in Atkin's and Elkies's improvements to Schoof's algorithm for counting points on elliptic curves [8]. Other applications include the computation of the endomorphism ring of an elliptic curve [25, 17] or of an isogeny between two elliptic curves [19].

More general modular polynomials occur in modern complex multiplication constructions for elliptic curves [14, 13, 11]; they are discussed in Section 4. When counting points on an elliptic curve by the Schoof–Elkies–Atkin algorithm, the modular polynomials are usually supposed to be available after a precomputation phase. This makes sense in situations where the algorithm is carried out many times for curves of about the same size, as is typical in cryptography. When establishing point counting records [12], however, one quickly realises that the computation of the modular polynomials itself becomes a bottleneck that may even dominate from a complexity theoretic point of view.

In this article, after surveying in Section 2 the state of the art as it appears in the literature, I present an algorithm based on evaluation and interpolation that computes modular polynomials in time essentially linear in the bit size of the polynomial. As it is based on floating point computations, one has to assume that rounding errors do not disturb the output, a heuristic that is supported by the implementation described in Section 5. A precise analysis of rounding errors to obtain a rigorously proved result appears to be out of reach. Note, however, that the correctness of the output may be checked probabilistically. For instance, one may instantiate the variable j occurring in the modular polynomial by the j-invariant of an elliptic curve over a finite field and verify that the specialised polynomial has the expected splitting behaviour.

Under the assumption that it is sufficient to use a floating point precision of $O(n)$ for a polynomial whose largest coefficient has n digits, the following holds:

Result 1.1 (heuristic). *Let $\Gamma' \subseteq \Gamma$ be a congruence subgroup, f a modular function for Γ' and $\Phi(X) \in \mathbb{C}(j)[X]$ the characteristic polynomial of f with respect to the function field extension $\mathbb{C}_{\Gamma'}/\mathbb{C}(j)$. Assume that Φ has coefficients in $\mathbb{Z}[j]$, and that the largest integer coefficient occurring in it has n digits. Suppose that a system of representatives of $\Gamma' \backslash \Gamma$ is known and that f can be evaluated at precision $O(n)$ in time $O(\log n \, M(n))$, where $M(n) \subseteq O\left(n \log^{1+\varepsilon}\right)$ is the time needed to multiply two numbers with n digits. Then there is an algorithm that computes Φ in time*

$$O\left(\deg_X \Phi \deg_j \Phi \left(\log^2 \max(\deg_X \Phi, \deg_j \Phi) + \log n\right) M(n)\right).$$

In particular, the classical modular polynomial Φ_ℓ such that $\Phi_\ell(j(z), j(\ell z)) = 0$ for ℓ prime is obtained in time

$$O\left(\ell^2 \log^2 \ell \, M(\ell \log \ell)\right) \subseteq O\left(\ell^3 \log^{4+\varepsilon} \ell\right).$$

If Φ is a dense polynomial and all of its coefficients (or at least a constant fraction of them) are of bit size about n, then this algorithm is linear in the size of the polynomial, except for the logarithmic factors.

The time bound of $O(\log n \, M(n))$ for evaluating modular functions at precision n is motivated by an algorithm due to Dupont; it relies on Newton iterations on an expression involving the arithmetic-geometric mean and reaches this complexity for a wide class of modular functions, including those built from Dedekind's η function and in particular k, k' and j [7, Theorem 5 and Section 7.2]. Alternatively, one may use an algorithm relying on multievaluation of q-expansions as described in [10, Section 6.3]. Its complexity, worse by a factor of $\log n$, is still sufficient for the running time of Result 1.1 to hold. The bound $M(n) \in O\left(n \log^{1+\varepsilon}\right)$ may be achieved by the classical Schönhage–Strassen algorithm [30] or the more recent and asymptotically faster algorithm due to Fürer [18].

The roots of Φ, that is, the algebraic conjugates of f, are given by the $f(M_\nu z)$ with $\Gamma' M_\nu$ running through a system of representatives of the residue classes $\Gamma' \backslash \Gamma$ (see [6]), so that

$$\Phi(X) = \prod_{\nu=1}^{[\Gamma:\Gamma']} (X - f(M_\nu z)) = X^{[\Gamma:\Gamma']} + \sum_{r=1}^{[\Gamma:\Gamma']} c_r X^{[\Gamma:\Gamma']-r}, \tag{1.1}$$

where the coefficients c_r of Φ are the elementary symmetric functions of the conjugates $f(M_\nu z)$.

By the q-expansion principle (cf. [6, §3]), a sufficient condition for the c_r to be polynomials in $\mathbb{Z}[j]$ is that f has rational integral q-coefficients, is holomorphic in $\mathbb{H} = \{z \in \mathbb{C} : \Im z > 0\}$ and all conjugates $f\left(\frac{az+b}{cz+d}\right)$ with $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma$ have integral algebraic q-coefficients.

2. Approaches to computing modular polynomials for $\Gamma^0(\ell)$

The case of $\Gamma^0(\ell)$ with ℓ prime, which is the most important one for applications, is also the simplest one from a theoretical point of view: The residue classes $\Gamma^0(\ell)\backslash\Gamma$ are represented by

$$T^\nu = \begin{pmatrix} 1 & \nu \\ 0 & 1 \end{pmatrix}, \nu = 0, \ldots, \ell - 1, \text{ and } S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

(The literature often concentrates on $\Gamma_0(\ell) = \begin{pmatrix} \ell & 0 \\ 0 & 1 \end{pmatrix}^{-1} \Gamma \begin{pmatrix} \ell & 0 \\ 0 & 1 \end{pmatrix} \cap \Gamma$. Since $\Gamma_0(\ell) = S^{-1}\Gamma^0(\ell)S$, a modular function f for $\Gamma^0(\ell)$ yields the function $f(Sz)$ for $\Gamma_0(\ell)$, and $f(Sz)$ is actually a conjugate of f. So both result in the same modular polynomial Φ. Or phrased differently: $\mathbb{C}_{\Gamma^0(\ell)}/\mathbb{C}_\Gamma$ is not normal, and another one of its embeddings is $\mathbb{C}_{\Gamma_0(\ell)}/\mathbb{C}_\Gamma$.)

Different functions have been suggested in the literature:

- $j(z/\ell)$, leading to the so-called "classical" or "traditional" modular polynomials;
- $\mathfrak{w}_\ell(z)^{2s} = \left(\frac{\eta(z/\ell)}{\eta(z)}\right)^{2s}$ for $s = 12/\gcd(12, \ell - 1)$, yielding the so-called "canonical" modular polynomials;
- functions that are moreover invariant under the Fricke–Atkin–Lehner involution $z \mapsto \frac{-\ell}{z}$; in particular, functions of the form

  $$\frac{T_r(\eta(z)\eta(\ell z))}{\eta(z)\eta(\ell z)}$$

  evaluated in $\frac{-1}{z}$ with T_r the r-th Hecke operator and linear combinations thereof as described in [27, Ch. 5.3].

All these functions lead to modular polynomials with coefficients in $\mathbb{Z}$.

2.1. **Heights of modular polynomials.** The (logarithmic) height of a modular polynomial with coefficients in $\mathbb{Z}$ is defined as the logarithm of the largest absolute value of the coefficients. It provides a lower bound on the arithmetic precision required to compute the polynomial. For the classical polynomial between $j(z)$ and $j(z/\ell)$, the height is given by

$$6(\ell+1)\,(\log \ell + O(1)) \subseteq O(\ell \log \ell)$$

according to [4]. For alternative modular polynomials, one observes a similar growth of the coefficients with ℓ, but the constant is usually smaller. It should be possible to adapt the argumentation in [4] to obtain a similar bound for further classes of modular polynomials.

2.2. **Using q-expansions.** The system of representatives of $\Gamma^0(\ell)\backslash\Gamma$ and thus the conjugates of the modular function f being explicitly known, one may use (1.1) to compute the modular polynomial Φ, if only a procedure for recognising its coefficients c_r as polynomials in $\mathbb{Z}[j]$ is available. The straightforward and most popular approach is to use q-expansions: The expansion of j is of the form $q^{-1} + \cdots$, so that knowing the expansion $c_r = c_{r,k} q^{-k} + \cdots$, one deduces that c_r has leading term $c_{r,k} j^k$ as a polynomial in j; one subtracts the q-expansion of this term and continues in the same vein. In fact, only the non-positive q-powers in the series expansions of the c_r are needed. Several variants of the algorithm are described in [26] with a brief complexity analysis in [8, Section 3]. We shall describe only the algorithm with the best complexity and give a detailed analysis of its running time.

Let f be a modular function for $\Gamma^0(\ell)$ with valuation $v < 0$ at infinity; that is, f admits a q-expansion of the form

$$f(z) = \sum_{k=v\ell}^{\infty} a_k q^{k/\ell} \in \mathbb{C}((q^{1/\ell})) \text{ with } q^{1/\ell} = e^{2\pi i z/\ell}.$$

Suppose that the a_k are rational integers. (For instance, $f = j(z/\ell)$ resp. $f = \mathfrak{w}_\ell^{2s}$ are valid choices with $v = -1/\ell$ resp. $v = -\frac{s(\ell-1)}{12\ell}$.) Then the q-expansions of the conjugates $f_\nu = f(T^\nu z)$ are of the same form with the coefficients a_k being multiplied by powers of $\zeta_\ell = e^{2\pi i/\ell}$.

The last conjugate $f_\infty = f(Sz)$ poses difficulties since it cannot be described generically, $q\left(\frac{-1}{z}\right) = q^{-2\pi i/z}$ being unrelated to q in a simple way, and is thus treated separately. (And it is the fact that there is only one of them that makes the case of $\Gamma' = \Gamma^0(\ell)$ stand out so that it is comparatively easily handled via the q-expansion approach.) Denote by v_∞ the order of f_∞ at infinity. For $f = j(z/\ell)$, one has $f_\infty = j(\ell z) = q^{-\ell} + \cdots$ and $v_\infty = -\ell$, and for $f = \mathfrak{w}_\ell^{2s}$, the transformation behaviour of η yields $f_\infty = \left(\sqrt{\ell}\,\frac{\eta(\ell z)}{\eta(z)}\right)^{2s} = \ell^s q^{s(\ell-1)/12} + \cdots$ with $v_\infty = \frac{s(\ell-1)}{12} = \frac{\ell-1}{\gcd(12,\ell-1)}$.

In particular, v_∞ may be positive or negative. If it is negative, the term with lowest q-exponent occurring in (1.1) is $f_\infty \cdot \prod_{\nu=0}^{\ell-1} f_\nu$ of exponent $\ell v + v_\infty$; so the degree of Φ in j is $|\ell v + v_\infty|$, reached for $c_{\ell+1}$. For instance, the degree in j is $\ell+1$ for the classical modular polynomial Φ_ℓ. If v_∞ is positive, the term with lowest q-exponent is $\prod_{\nu=0}^{\ell-1} f_\nu$, so the degree $|\ell v|$ of Φ in j is reached in c_ℓ. For instance, the degree in j of the canonical modular polynomials is $\frac{s(\ell-1)}{12}$.

We first consider the complexity of determining the partial product

$$\prod_{\nu=0}^{\ell-1}(X-f_\nu) = X^\ell + \sum_{r=1}^{\ell} \tilde{c}_r X^{\ell-r} \in \mathbb{Z}[\zeta_\ell]((q^{1/\ell}))[X].$$

One may proceed by computing the Newton sums $s_r = \sum_{\nu=0}^{\ell-1} f_\nu^r$ for $1 \le r \le \ell$ and use Newton's recurrence formulæ to deduce the $\tilde{c}_r$. Following [26], one obtains with $f^r = \sum_{k=rv\ell}^{\infty} a_{k,r} q^{k/\ell}$ that

$$s_r = \sum_{k=\lceil rv \rceil}^{\infty} \ell\, a_{\ell k,r}\, q^k \in \mathbb{Z}((q)), \tag{2.1}$$

so that in fact the roots of unity do not interfere with the computations. One sees that the valuations of the s_r at infinity are bounded below by $\lceil rv \rceil$ (and in general, they will be equal to it). Assume that $d+1$ terms of them are known, that is, s_r is known up to and including the coefficient of $q^{\lceil rv \rceil + d}$. Then one shows by induction on Newton's relation

$$\tilde{c}_r = \frac{1}{r}\sum_{k=1}^{r}(-1)^{k+1} s_k\, \tilde{c}_{r-k} \tag{2.2}$$

that the valuation of $\tilde{c}_r$ is also bounded below by $\lceil rv \rceil$ and that it is also known up to $q^{\lceil rv \rceil + d}$.

If $v_\infty \ge 0$, we need only the terms of the $\tilde{c}_r$ with non-positive exponents; for $v_\infty < 0$, we loose some precision, in fact $|v_\infty|$ terms, through the final multiplication by $X - f_\infty$. Thus, the value required for d is given by

$$d = \ell|v| + \max(0, -v_\infty) = \deg_j \Phi.$$

This is an indication that the algorithm given above should be optimal among those relying on q-expansions: Outputting polynomials with up to $\deg_j \Phi + 1$ coefficients, it manipulates series with as many terms. Unfortunately, the computation of the s_r involves decimating the series for f^r by ℓ. Eventually, a factor of ℓ is lost in the running time since $O(\ell d)$ terms of the f^r are needed.

Once the q-coefficients of f are known, the f^r and s_r are thus computed by $O(\ell)$ multiplications of series with $O(\ell d)$ terms and the $\tilde{c}_r$ by $O(\ell^2)$ multiplications of series with $O(d)$ terms. Let $M_q(d)$ be the number of arithmetic operations in $\mathbb{Z}$ required to multiply two dense q-expansions with d terms. As M_q is at least linear, the total complexity of the series computations becomes

$$O(\ell\, M_q(\ell d)).$$

For functions f related to η quotients, such as $f = \mathfrak{w}_\ell^{2s}$ or $f = j(z/\ell) = \frac{\left(\mathfrak{w}_2^{24}(z/\ell)+16\right)^3}{\mathfrak{w}_2^{24}(z/\ell)}$, the effort for computing their q-expansions is negligible, since it corresponds to a constant number of arithmetic operations with series starting from the easily written down series expansion of η.

To write the c_r as polynomials in j, one has to compute the non-positive parts of the q-expansions of the j^k for $1 \le k \le d$. This can be done in time $O(d\, M_q(d))$, which is negligible compared to the previous steps because $d \in O(\ell)$. Identifying the c_r as polynomials then corresponds to solving triangular systems of linear equations, requiring altogether $O\left(\ell d^2\right)$ operations with integers.

The total complexity thus becomes

$$O\left(\ell\, M_q(\ell^2)\right)$$

integer operations.

Let $n \in O(\ell \log \ell)$ as in Section 2.1 be a bound on the height of the modular polynomial. Then the bit complexity of the algorithm is given by

$$O\left(\ell^3 \log \ell\, M(n)\right) \subseteq O\left(\ell^4 \log^{3+\varepsilon} \ell\right)$$

In [3, Appendix A] it is argued that in fact the running time is higher, on grounds that the coefficients of the powers of j grow faster than $O(\ell \log \ell)$. However, this objection can be dismissed by carrying out all computations modulo a sufficiently large prime of bit size $n \in O(\ell \log \ell)$. Alternatively, and probably preferably in practice, one may work modulo small primes and use the Chinese remainder theorem. Both approaches yield the desired complexity.

Remark. If Φ_ℓ is sought only modulo some prime p, then the complete algorithm can be carried out modulo p as soon as p is larger than ℓ to make the divisions in (2.2) possible. The bit complexity becomes

$$O\left(\ell^3 \log \ell \log p\, (\log \log p)^{1+\varepsilon}\right).$$

2.3. **Charles–Lauter.** In [3], the authors describe an algorithm to compute the classical modular polynomial Φ_ℓ directly modulo a prime p without recourse to q-expansions of modular functions. Instead, they rely on the moduli interpretation of Φ_ℓ, which characterises pairs of ℓ-isogenous elliptic curves. So the algorithm manipulates only elliptic curves and explicit isogenies over extension fields of $\mathbb{F}_p$.

Its basic building block is the computation of the instantiated polynomial $\overline{\Phi} = \Phi_\ell(X, \overline{j})$ with $\overline{j} \in \mathbb{F}_{p^2}$ the j-invariant of a supersingular elliptic curve E. The roots of $\overline{\Phi}$ are the j-invariants of the $\ell + 1$ curves that are ℓ-isogenous to E, and that may be obtained via Vlu's formulæ [31] once the complete ℓ-torsion of E is known. These are ℓ^2 points defined over an extension of $\mathbb{F}_{p^2}$ of degree $O(\ell)$ in the supersingular case; in fact, one expects a degree of $\Theta(\ell)$ virtually all of the time. Thus writing down the ℓ-torsion points requires a time of $\Omega(\ell^3 \log p)$.

After having repeated the procedure for $\deg_j \Phi_\ell = \Theta(\ell)$ different values of $\overline{j}$, one obtains the coefficients of Φ_ℓ modulo p by interpolation. (This is analogous to the floating point approach of Section 3, and more details are given there.) The complexity of the algorithm is thus at least

$$\Omega\left(\ell^4 \log p\right),$$

and an upper bound of $O\left(\ell^{4+\varepsilon} \log^{2+\varepsilon} p + \log^{4+\varepsilon} p\right)$ is proved in [3, Theorem 3.2] under the generalised Riemann hypothesis.

Hence, this algorithm is slower by a factor of order ℓ than the one presented in Section 2.2 relying on q-expansions. This is due to the costly determination of isogenies via their kernels: While the isogenies are defined over $\mathbb{F}_{p^2}$, their kernels lie in an extension of degree $O(\ell)$. The q-expansions of the conjugates of $j(z/\ell)$, on the other hand, provide a synthetic description of the curves that are ℓ-isogenous to the one with j-invariant $j(z)$, and using them it is sufficient to carry out all computations in $\mathbb{F}_p$ in order to obtain $\Phi_\ell \bmod p$.

3. Evaluation–interpolation

The approach for calculating modular polynomials that is described in this section is in fact neither new nor particularly involved. It has been suggested to me by R. Schertz during our work on the class invariants of [14], and later R. Dupont pointed out to me that it can already be found in [1, Chapter 4.5]. However, it has not been noticed before that the algorithm allows to lower the exponent of the computational complexity by 1 and thus to reach an essentially optimal complexity if fast polynomial arithmetic and fast techniques for evaluating modular functions as described in [7, 10] are used.

The basic idea of the evaluation–interpolation approach is to specialise and to compute the identity (1.1) between modular functions in several complex floating point arguments (the evaluation phase); and then to interpolate the coefficients in order to recognise them as polynomials in j. Precisely, (1.1) can be rewritten as

$$\Phi(X, j(z)) = \prod_{\nu=1}^{[\Gamma:\Gamma']} (X - f(M_\nu z)) = X^{[\Gamma:\Gamma']} + \sum_{r=1}^{[\Gamma:\Gamma']} c_r(z) X^{[\Gamma:\Gamma']-r} \tag{3.1}$$

for all z in the upper complex half plane.

Evaluating the conjugates $f(M_\nu z)$ of f in a number of complex arguments z_k with $\Im z_k > 0$, multiplying out the left hand side as a polynomial in $\mathbb{C}[X]$ and separating the coefficients according to powers of X yields the values $c_r(z_k)$. Writing $c_r = \sum_{s=0}^{\deg_j \Phi} c_{r,s} j^{\deg_j \Phi - s}$, the $c_r(z_k)$ are actually the values of the polynomial c_r in $j(z_k)$, so that the coefficients $c_{r,s} \in \mathbb{C}$ can be retrieved by interpolation as soon as the $c_r(z_k)$ are known for $\deg_j \Phi + 1$ values of z_k. The final step is to round the $c_{r,s}$ to rational integers. The degree of the modular polynomial in j or an upper bound on it may be known beforehand; if it is not, then the algorithm can be repeated with increasing guesses for $\deg_j \Phi$ until rounding to integers succeeds (and doubling the guess at each time results in the same asymptotic complexity as taking the correct value from the beginning).

Let $E(n)$ be the bit complexity for evaluating the modular functions f or j at a precision of n bits (here, f is considered to be fixed, while n tends to infinity with ℓ). Then the evaluation phase requires $(\ell + 1)(\deg_j \Phi + 1)$ evaluations of f at a cost of

$$O\left(\ell \deg_j \Phi \, E(n)\right)$$

and the reconstruction of $\deg_j \Phi + 1$ polynomials of degree $\ell + 1$ from their roots. Multiplying complex polynomials by the FFT, this reconstruction step takes

$$O\left(\deg_j \Phi \, \ell \log^2 \ell \, M(n) + \log n \, M(n)\right),$$

where the (eventually negligible) term $\log n \, M(n)$ accounts for the computation of a primitive root of unity of sufficiently high order to carry out all the FFTs, and as usual $M(n)$ is the time needed to multiply two n-bit numbers.

The interpolation phase consists of ℓ interpolations of polynomials of degree $\deg_j \Phi$. Employing again fast algorithms such as [20, Algorithm 10.11], this takes

$$O\left(\ell \deg_j \Phi \log^2(\deg_j \Phi) \, M(n)\right),$$

once the roots of unity are available.

It is shown in [7] that among others, Dedekind's η function can be evaluated at precision n in time $E(n) \in O(\log n \, M(n))$ uniformly for the argument in $\mathcal{F} = \left\{ z \in \mathbb{H} : |\Re z| \leq \frac{1}{2}, |z| \geq 1 \right\}$. (The restriction to $\mathcal{F}$ is not crucial since we may freely choose our interpolation points.) So in particular, all functions built from η such as j and $\mathfrak{w}_\ell$ satisfy $E(n) \in O\left(\log n \, M(n)\right)$. (Alternatively, one may use multievaluation as described in [10, Section 6.3] with a slightly worse amortised cost of $O\left(\log^2 n \, M(n)\right)$ per value, which still allows to reach the final complexity of Result 1.1.)

In total, the steps add up to a running time of

$$\begin{aligned} & O\left((\ell \deg_j \Phi \log^2 \max(\ell, \deg_j \Phi) + \log n) M(n)\right) \\ \subseteq \; & O\left((\deg_X \Phi \deg_j \Phi \log^2 \max(\deg_X \Phi, \deg_j \Phi) + \log n) M(n)\right), \end{aligned}$$

and the latter formulation is valid for arbitrary congruence subgroups Γ' in the place of $\Gamma^0(\ell)$.

Here, n must be at least as large as the logarithmic height of Φ, and maybe bigger to account for rounding errors. In the case of the classical modular polynomial Φ_ℓ the bound $O(\ell \log \ell)$ of Section 2.1 yields a complexity of

$$O\left(\ell^2 \log^2 \ell M(\ell \log \ell)\right).$$

This proves Result 1.1.

4. Further kinds of modular polynomials

4.1. **Modular functions of composite level.** Besides its better complexity compared to the approach using q-expansions, the evaluation–interpolation algorithm has the advantage of a great flexibility, which makes it easily adaptable to a variety of different modular polynomials. In fact, the proof of Result 1.1 in Section 3 does not use special properties of $\Gamma^0(\ell)$ and is valid for any congruence subgroup.

An application is given by the computation of the polynomial relationship between j and a modular function f of composite level N. At first sight, these polynomials are not of great interest. In the point counting or more generally isogeny computation context, it is more efficient to express an isogeny of composite degree as a composition of prime degree isogenies. But this kind of modular polynomials occurs naturally in the context of [14], in which elliptic curves with complex multiplication are obtained via modular functions f of composite level. Levels $N = pq$ or $N = p^2$ with p and q prime are examined in [14], but more general settings can be devised in a straightforward way. The polynomial relationship between f and j is used to derive from a special value of f the j-invariant of the corresponding elliptic curve.

If the q-expansion approach were to be pursued to compute this kind of modular polynomials, it would be necessary to somehow obtain the q-expansions of the conjugates $f(M_\nu z)$ for a system of representatives (M_ν) of $\Gamma' \backslash \Gamma$. This is straightforward only for translations; for all other matrices, it is necessary to take the transformation behaviour of f under unimodular matrices into account, which requires ad hoc computations for each particular function. (This is illustrated by $j(Sz/\ell) = j(\ell z)$ and $\mathfrak{w}_\ell^{2s}(Sz) = \ell^s \left(\frac{\eta(\ell z)}{\eta(z)}\right)^{2s}$, which are not derived from $j(z/\ell)$ resp. $\mathfrak{w}_\ell(z)$ by the same generic transformation.) In the case of $\Gamma^0(\ell)$, only the matrix S asks for special treatment; in the case of composite level, many more special matrices appear. For instance, the full article [15] is concerned with deriving the conjugates of only the functions $\left(\frac{\eta(z/p_1)\eta(z/p_2)}{\eta(z/(p_1p_2))\eta(z)}\right)^s$ for p_1, p_2 prime and $s = 24/\gcd(24, (p_1-1)(p_2-1))$.

In the evaluation–interpolation approach, the only difference between $\Gamma^0(\ell)$ and other congruence subgroups Γ' is the enumeration of the system of representatives (M_ν). For most interesting Γ' (and in particular for arbitrary $\Gamma^0(N)$), this is easy; in any case, this step is independent of the particular function f. Then Result 1.1 applies, and one obtains an algorithm that is essentially linear in its output size.

4.2. **Schläfli equations.** In [29], Schläfli examines transformations of prime level ℓ of special modular functions different from j that lead to particularly simple modular equations. Weber gives a systematic treatment of them in [32, §§73–74]. Let $\mathfrak{f} = \zeta_{48}^{-1}\frac{\eta((z+1)/2)}{\eta(z)} = \zeta_{48}\mathfrak{w}_2(z+1)$ be one of "the Weber functions", a modular function of level 48. Let ℓ be a prime not dividing this level. Then $g(z) = \mathfrak{f}(z/\ell)$ is the root of a monic polynomial $\Phi_\ell^{\mathfrak{f}}(X)$ of degree $\ell+1$ with coefficients in $\mathbb{Z}[\mathfrak{f}]$.

The evaluation–interpolation approach allows to easily obtain this polynomial with only minimal modifications to the algorithm. The function $\mathfrak{f}$ being modular for a subgroup of $\Gamma(48)$ and ℓ being different from 2 and 3, a quick computation reveals that g is modular for $\Gamma(48)\cap\Gamma^0(\ell)$. The polynomial $\Phi_\ell^{\mathfrak{f}}$ is thus given by an equation analogous to (1.1):

$$\Phi_\ell^{\mathfrak{f}}(X) = \prod_{\nu=1}^{\ell+1}(X - g(M_\nu z)) = X^{\ell+1} + \sum_{r=1}^{\ell} c_r X^{\ell+1-r}, \tag{4.1}$$

where the M_ν range over a set of representatives of $(\Gamma(48)\cap\Gamma^0(\ell))\backslash\Gamma(48)$. Such a set may be obtained by multiplying the standard representatives of $\Gamma^0(\ell)\backslash\Gamma$ from the left by a matrix in $\Gamma^0(\ell)$ such that they end up in $\Gamma(48)$. Precisely, a possible set of representatives is given by $\begin{pmatrix}1 & 48\nu\\ 0 & 1\end{pmatrix}$ for $\nu = 0,\dots,\ell-1$ (corresponding to the translations) and $\begin{pmatrix}1-48k & 48k\\ -48k & 1+48k\end{pmatrix}$ with $k = 48^{-1} \bmod \ell$, corresponding to the inversion S.

So the only modification required for the evaluation–interpolation algorithm to work is this adaptation of the matrices M_ν, and of course j has to be replaced by $\mathfrak{f}$ in the interpolation phase to recover the coefficients c_r as elements of $\mathbb{Z}[\mathfrak{f}]$.

Hence Result 1.1 clearly applies also to the Schläfli equations, showing that they can be computed in essentially linear time with respect to the output size.

Taking specifics of the function $\mathfrak{f}$ into account, a more efficient algorithm can be obtained, gaining a constant factor. Weber shows in [32, p. 266] that only every 24-th coefficient in $\mathfrak{f}$ of the c_r may be non-zero. Precisely, $\mathfrak{f}^i g^k$ having a non-zero coefficient implies $\ell i + k \equiv \ell + 1 \pmod{24}$. Hence the number of evaluations can be reduced by a factor of about 24. (In [1, Ch. 4.5], similar sparse modular equations are studied, in which one out of eight coefficients is non-zero. The Borwein's suggest in this chapter the evaluation–interpolation approach while profiting of this sparseness.) Weber shows that the $\Phi_\ell^{\mathfrak{f}}$ are symmetric, which could also be taken into account during the interpolation phase.

Weber derives the Schläfli equations in a very compact form. He considers the new functions
$$A = \left(\frac{\mathfrak{f}}{g}\right)^r + \left(\frac{g}{\mathfrak{f}}\right)^r \text{ and } B = (\mathfrak{f}g)^s + \left(\frac{2}{\ell}\right)\frac{2^s}{(\mathfrak{f}g)^s}$$
with r, s such that $12|(\ell-1)r$, $12|(\ell+1)s$ and $\frac{(\ell-1)r}{12} \equiv \frac{(\ell+1)s}{12} \pmod 2$. Depending on $\ell \bmod 24$, these conditions are satisfied by some r and s such that $2r|\ell+1$ and $2s|\ell-1$. Weber [32, p. 268] then shows that
$$\Phi_\ell^{\mathfrak{f}} = (\mathfrak{f}g)^{(\ell+1)/2}\left(A^{(\ell+1)/(2r)} - B^{(\ell-1)/(2s)} + \sum_{\alpha=0}^{(\ell+1)/(2r)-1}\sum_{\beta=0}^{(\ell-1)/(2s)-1} c_{\alpha,\beta}A^\alpha B^\beta\right).$$
In this form, the polynomial has about $\ell^2/(4rs)$ coefficients, which is often less than the roughly $\ell^2/24$ coefficients if it is written as an equation between $\mathfrak{f}$ and g. However, the more compact form appears to be less suited to the evaluation–interpolation algorithm: There is no easy way, in the spirit of (1.1), of obtaining its values when interpreting it as a univariate polynomial in $A(z)$ with coefficients in $\mathbb{Z}[B(z)]$. So instead of performing $O(\ell)$ interpolations of polynomials of degree $O(\ell)$, one would apparently have to obtain all coefficients at once by solving a linear system with $O(\ell^2)$ unknowns, resulting in a complexity that is worse than that of Result 1.1.

4.3. **Generalised Schläfli equations.** The Schläfli equations of the previous paragraph can obviously be generalised to further modular functions: Given a modular function f and a prime ℓ, one may want to compute the polynomial relating $f(z)$ and $f(z/\ell)$.

In [24, Ch. 5], a theory analogous to Weber's is developped for the functions $\mathfrak{w}_\ell$. The derived polynomials are used to obtain symbolically minimal polynomials of special algebraic values of η quotients.

In the context of a p-adic algorithm for the computation of class polynomials and ultimately elliptic curves with complex multiplications, another application is developed in [2, Ch. 6.8]. Let τ be a quadratic integer. Then the singular value $j(\tau)$ lies in the ring class field for the order $\mathcal{O} = [1, \tau]_\mathbb{Z}$ and is the j-invariant of an elliptic curve with complex multiplication by $\mathcal{O}$. In [5], the authors describe an algorithm to compute the canonical lift of $j(\tau)$ to the p-adic numbers at arbitrary precision by some kind of Newton iteration. The main algorithmic step consists of applying an isogeny to $j(\tau)$ (or more precisely to the elliptic curve with this j-invariant) that corresponds to a smooth principal ideal and that can thus be realised by composing isogenies of manageable prime degrees. In [2], the algorithm is generalised to other class invariants. Let f be a modular function for $\Gamma^0(N)$ such that $f(\tau)$ is a class invariant in the sense that it also lies in the ring class field of $\mathcal{O}$. Let ℓ be a prime number not dividing N that splits as $(\ell) = \mathfrak{l}\bar{\mathfrak{l}}$ in $\mathcal{O}$. As in the case of $j(\tau)$, one needs to "apply an isogeny" and compute $f(\mathfrak{l}^{-1})$. This value is given as a root of the modular polynomial Ψ between $f(z)$ and $f(z/\ell)$, specialised with $f(\tau)$ in the place of $f(z)$. (The modular polynomial Φ between $f(z)$ and $j(z)$, treated in Section 4.1, also plays a role as it helps to choose whenever Ψ has multiple roots: $f(\mathfrak{l}^{-1})$ is also a root of Φ specialised with $j(\mathfrak{l}^{-1})$ in the place of $j(z)$.)

A quick computation shows that $g(z) = f(z/\ell)$ is modular for $\Gamma^0(\ell) \cap \Gamma^0(N) = \Gamma^0(\ell N)$, so that (4.1) yields the minimal polynomial of g over $\mathbb{C}_{\Gamma^0(N)}$ if the M_ν are chosen as a system of representatives of $\Gamma^0(\ell N)\backslash\Gamma^0(N)$. Such a system is obtained precisely as for the Schläfli equations by multiplying the standard representatives of $\Gamma^0(\ell)\backslash\Gamma$ by matrices in $\Gamma^0(\ell)$ so that they end up in $\Gamma^0(N)$. For instance, a set of representatives is given by $\begin{pmatrix} 1 & N\nu \\ 0 & 1 \end{pmatrix}$ for $\nu = 0, \ldots, \ell - 1$ and $\begin{pmatrix} 1 - Nk & Nk \\ -Nk & 1 + Nk \end{pmatrix}$ with $k = N^{-1} \bmod \ell$. So at first sight, the evaluation–interpolation algorithm seems to apply.

The problem lies in recognising the coefficients c_r as polynomials in f. In fact, the c_r are modular for $\Gamma^0(N)$, so unless $C_{\Gamma^0(N)} = \mathbb{C}(f)$ (otherwise said, f is a *Hauptmodul* for $\Gamma^0(N)$), there is no reason that they are rational in f. Generically, one expects $C_{\Gamma^0(N)} = \mathbb{C}(j, f)$ and may hope (under suitable integrality and rationality conditions) to recover the c_r as elements of $\mathbb{Z}[j, f]$.

A necessary condition for f being a Hauptmodul is that the modular curve $X_0(N)$ is of genus 0, which limits the possible values of N to a very short list. In general, it will be necessary to replace the minimal polynomial of g over $\mathbb{C}_{\Gamma^0(N)}$ by that over $\mathbb{C}(f)$; its degree will be $\ell + 1$ multiplied by $[\mathbb{C}_{\Gamma^0(N)} : \mathbb{C}(f)]$. It is not clear whether the evaluation–interpolation approach can be adapted to this context, since $\mathbb{C}(f)$ is not the field of modular functions for any congruence subgroup.

5. Implementation

The evaluation–interpolation algorithm has been implemented in C by the author for different kinds of modular polynomials. One big advantage of the algorithm (besides its superior complexity) is that it is rather straightforward to implement, especially in the context of complex multiplication (cf. [10]), where the major building blocks such as the evaluation of modular functions and computations with polynomials over floating point numbers are already in use. The author's implementation relies on the existing libraries `gmp` [22] with an assembly patch for AMD64 by P. Gaudry [21] for fast basic multiprecision arithmetic, `mpfr` [23] and `mpc` [16], which provide elementary operations with arbitrary precision floating point real and complex numbers with exact rounding. A library for fast polynomial operations via Karatsuba, Toom–Cook and the FFT has been written for the occasion [9]. All timings mentioned in the following have been obtained on Opteron processors clocked at 2.4 GHz. The heights are given as logarithms in base 2, and the arithmetic precision is given in bits.

5.1. Details of the implementation.

5.1.1. *Details slowing the implementation down asymptotically.* Of the different techniques for evaluating modular functions described in [10], the asymptotically fast ones are not beneficial in the present context. For computing univariate class polynomials, they do not pay off at degrees below 100 000. Such high degrees are for the time unachievable for bivariate modular polynomials, since the number of coefficients would make it impossible to store the result. Hence all examples have been computed via the sparse representation of the η function as a q-series.

The interpolation phase turns out to take negligible time; thus a simple quadratic algorithm using iterated differences is employed instead of a quasilinear one.

5.1.2. *Details speeding the implementation up.* In order to work with real instead of complex numbers as much as possible, purely imaginary values are chosen for the z_k. Then the $q(z_k)$ are real, and so are the values $j(z_k)$ and thus, by (3.1), the

$\Phi(X, j(z_k))$. While the values of the conjugates $f(M_\nu z_k)$ still have to be computed as complex numbers, roughly half of them suffice for functions f for $\Gamma^0(\ell)$ with rational q-coefficients: $f(z_k)$ is real, $f(z_k - r)$ is the complex conjugate of $f(z_k + r)$ for $r \in \mathbb{Z}$, and finally $f(-1/z_k)$ has to be real again to obtain a real polynomial in the end. When reconstructing $\Phi(X, j(z_k))$ from its roots, the complex conjugate roots can be grouped so that only real arithmetic is required.

Also the interpolation phase uses only real numbers. It is furthermore helped by choosing the z_k such that the $j(z_k)$ lie in a simple arithmetic progression, for instance, $j(z_k) = 1728 + k$ for $k \geq 1$.

5.1.3. *Parallelisation.* Another nice feature of the evaluation–interpolation algorithm is that it can be parallelised almost arbitrarily, in principal up to distributing at the level of each modular function evaluation. A more natural, coarser parallelisation of the evaluation phase, which needs almost no communication, is amply sufficient in practice: Each processor treats a different z_k and computes an approximation $\Phi(X, j(z_k))$ by evaluating the conjugates $(f(M_\nu z_k))_\nu$ and reconstructing the polynomial from these roots. The result is written into a file on a shared hard disk. The interpolation phase has been too fast to warrant a parallel implementation. If it were to become a bottleneck, a natural way of distributing the work would be to have each processor compute the coefficients of a different polynomial $c_r \in \mathbb{Z}[j]$.

Implementing the communication via files solves a second problem. During the evaluation phase, the matrix $(c_r(z_k))_{k,r}$ is computed row-wise; for the interpolation, it is accessed column-wise. However, the matrix is roughly as big as the final modular polynomial and for the largest computed examples (see Section 5.2.1) does not fit into main memory any more. Writing each row into a different file during the evaluation and reading all files in parallel during the interpolation can be seen as a means of transposing the matrix on disk without having to keep it in main memory.

5.1.4. *Arithmetic precision.* Except for the classical modular polynomials, no upper bound on the height of the polynomials is known, and even in the classical case, the bound is not completely explicit, but contains an $O(1)$ term (see Section 2.1). In the implementation, an approximate bound is obtained by carrying out all computations first at very low precision (100 bits). Due to the numeric instability of interpolation, the resulting coefficients are wrong already in the first digit, but one obtains an idea of the height of the correct polynomial (actually, the height at low precision turns out to be larger than the real one). This estimate is multiplied by a factor (between 1.1 and 2 depending on the function and the level), determined experimentally, to deal with rounding errors.

In the context of class polynomial computations, it has been observed in [10] that virtually no rounding errors occur: Evaluating modular functions via the q-development of η as well as multiplication of polynomials is rather uncritical from a numerical point of view. It suffices to add a few bits to the target precision. This should also hold for the evaluation phase in our context. In practice, it can be observed, however, that a significantly higher precision is needed than just the height of the result. This can only be due to the interpolation phase, which implicitly handles Vandermonde matrices, that are known to be ill conditioned. It is an open problem to choose the interpolation points $j(z_k)$ so as to minimise the rounding errors.

5.2. Examples.

5.2.1. *Modular polynomials for* $\Gamma^0(\ell)$. For achieving the point counting record for elliptic curves with the Schoof–Elkies–Atkin algorithm [12], the polynomials have been systematically computed with the function

$$f_{\ell,r} = \frac{T_r(\eta(z)\eta(\ell z))}{\eta(z)\eta(\ell z)} \tag{5.1}$$

(evaluated in $-1/z$) suggested in [27]. Here,

$$T_r(f) = \frac{1}{r}\sum_{\nu=0}^{r-1} f\left(\frac{z+24\nu}{r}\right) + f(rz)$$

is the r-th Hecke operator for a prime $r \geq 5$ such that $24|(r-1)(\ell+1)$, r is a quadratic residue modulo ℓ and ℓ is a quadratic residue modulo r. (The unusual factor 24 in front of ν takes care of the fact that the exponents in the q-expansion of η are not integral due to the factor $q^{1/24}$; it eliminates 24-th roots of unity.)

It is not advisable to obtain the values of $f_{\ell,r}$ via its q-expansion. First of all, the expansion is dense and thus much slower to evaluate than that of η. But more importantly, q tends to infinity when z approaches the cusp 0 of the fundamental domain for $\Gamma^0(\ell)$, so that the q-series converges arbitrarily slowly. This is not an issue when relying on the evaluation of η only, since its known transformation behaviour under unimodular matrices allows to transform the arguments into the fundamental domain for Γ, whose only cusp is at infinity and corresponds to $q = 0$. So the implementation evaluates (5.1) numerically with $2r + 4$ evaluations of η per value of $f_{\ell,r}$. The resulting dependence of the running time on r can be seen clearly in the following examples.

ℓ	2039
r	5
degree in j	136
estimated height	5816
precision	6397
height	5040
time for evaluation	5900 s
time for interpolation	110 s

ℓ	2017
r	61
degree in j	156
estimated height	6211
precision	6832
height	5311
time for evaluation	74000 s
time for interpolation	130 s

The largest computed polynomial has a level of 10079. The computation takes less than two weeks on a small cluster with ten processors; as a compressed text file, the polynomial fills about 16 Gb of space. This illustrates that the limiting factor for the algorithm becomes not so much its running time, but rather the space requirements of its output, as can be expected for algorithms that have an essentially linear time complexity with respect to their output size.

ℓ	10079
r	5
degree in j	672
estimated height	31865
precision	35051
height	28825
time for evaluation	10 000 000 s $\approx$ 120 d
time for interpolation	56 000 s $\approx$ 16 h

It would be natural to try to linearly combine functions $f_{\ell,r}$ with different r to reduce the pole order at infinity, or even to find a function on $X_0(\ell)$ with minimal pole order. As a result, the degree of the modular polynomial in j (and thus the number of interpolation points) and the size of its coefficients (and thus the required precision) could be reduced. A high price, however, would have to be paid by an increased running time for the evaluation. As high performance clusters are becoming increasingly available, while bandwidth and storage space appear to be the bottleneck for modular polynomial computation, this might, however, be the road to follow for constructing a manageable database of modular polynomials up to a high level.

5.2.2. *Schläfli equations.* As explained in Section 4.2, a trivial modification to the enumeration of the $\ell + 1$ matrices M_ν adapts the code to computing the Schläfli equations between $\mathfrak{f}(z)$ and $\mathfrak{f}(z/\ell)$. It is then imperative to change the interpolation code so as to take advantage of the sparseness of the polynomial, in which only every 24-th coefficient is non-zero. This reduces the number of evaluations and (after a suitable transformation) the degree of the interpolated polynomials roughly by a factor of 24. The latter is also important since the interpolation phase becomes numerically more stable.

ℓ	2039
number of interpolation points	86
estimated height	2829
precision	3111
height	2380
time for evaluation	230 s
time for interpolation	35 s

5.2.3. *Generalised Schläfli equations.* As argued in Section 4.3, the minimal polynomial of $g(z) = f(z/\ell)$ over $\mathbb{C}(f)$ for some function f for $\Gamma^0(N)$ need not be of degree $\ell + 1$ any more, in which case it is a priori unclear how to obtain it by evaluation–interpolation. But besides the functions f that are Hauptmoduln for $\Gamma^0(N)$, a few others may be handled by this approach. Namely, let

$$\mathfrak{w}_{p_1,p_2} = \frac{\eta(z/p_1)\eta(z/p_2)}{\eta(z)\eta(z/(p_1p_2))}$$

with p_1, p_2 prime and $24|(p_1-1)(p_2-1)$ be the functions of level $N = p_1p_2$ suggested as class invariants in [14]. These functions are invariant under the Fricke–Atkin–Lehner involution. Now, $X_0^+(N)$ is of genus 0 for $N = 35$ and $N = 39$ [28], and apparently, the double η quotients are Hauptmoduln in this case. The degree of the modular equation of transformation level ℓ thus remains $\ell + 1$ (an observation made for $N = 35$ in [2, Ch. 7.4]).

For instance, with $f = \mathfrak{w}_{3,13}$ and $g(z) = f(z/\ell)$, one obtains the following polynomials $\Phi_\ell^{\mathfrak{w}_{3,13}}$:

$$\begin{aligned}
\Phi_2^{\mathfrak{w}_{3,13}} &= g^3 + f^3 - g^2f^2 - gf + 2(g^2f + fg^2) \\
\Phi_5^{\mathfrak{w}_{3,13}} &= g^6 + f^6 - g^5f^5 - gf + 35g^3f^3 \\
&\quad +5(g^5f^4 + g^4f^5 + g^5f + gf^5 - g^4f^3 - g^3f^4 - g^3f^2g^2f^3 + g^2f + gf^2) \\
&\quad +10(-g^5f^2 - g^2f^5 + g^4f^4 + g^4f^2 + g^2f^4 - g^4f - gf^4 + g^2f^2) \\
\Phi_7^{\mathfrak{w}_{3,13}} &= g^8 + f^8 - g^7f^7 - gf \\
&\quad +7(g^7f^6 + g^6f^7 - g^7f^5 - g^5f^7 + g^6f^4 + g^4f^6 + g^6f^2 + g^2f^6) \\
&\quad +7(g^4f^2 + g^2f^4 - g^3f - gf^3 + g^2f + gf^2) \\
&\quad +14(-g^7f - gf^7 + g^5f^4 + g^4f^5 + g^5f^3 + g^3f^5 + g^4f^3 + g^3f^4) \\
&\quad +21(-g^7f^4 - g^4f^7 + g^7f^3 + g^3f^7 - g^6f^5 - g^5f^6 + g^6f^3 + g^3f^6) \\
&\quad +21(g^5f^2 + g^2f^5 + g^5f + gf^5 - g^4f - gf^4 - g^3f^2 - g^2f^3) \\
&\quad +42(g^6f^6 + g^2f^2) - 182g^4f^4
\end{aligned}$$

Like the Schläfli equations, the polynomials are symmetric in f and g, but as can be seen from these and further examples, they are no more sparse.

References

1. Jonathan M. Borwein and Peter B. Borwein, *Pi and the AGM*, Wiley, New York, 1987.
2. Reinier Bröker, *Constructing elliptic curves of prescribed order*, Proefschrift, Universiteit Leiden, 2006.
3. Denis Charles and Kristin Lauter, *Computing modular polynomials*, LMS Journal of Computation and Mathematics **8** (2005), 195–204.
4. Paula Cohen, *On the coefficients of the transformation polynomials for the elliptic modular function*, Mathematical Proceedings of the Cambridge Philosophical Society **95** (1984), 389–402.
5. Jean-Marc Couveignes and Thierry Henocq, *Action of modular correspondences around CM points*, Algorithmic Number Theory — ANTS-V (Berlin) (Claus Fieker and David R. Kohel, eds.), Lecture Notes in Computer Science, vol. 2369, Springer-Verlag, 2002, pp. 234–243.
6. Max Deuring, *Die Klassenkörper der komplexen Multiplikation*, Enzyklop. d. math. Wissenschaften, vol. I 2 Heft 10, Teubner, Stuttgart, 2e ed., 1958.
7. Régis Dupont, *Fast evaluation of modular functions using Newton iterations and the AGM*, To appear in *Mathematics of Computation*, `http://www.lix.polytechnique.fr/Labo/Regis.Dupont/preprints/Dupont_Fast%EvalMod.ps.gz`, 2007.
8. Noam D. Elkies, *Elliptic and modular curves over finite fields and related computational issues*, Computational Perspectives on Number Theory: Proceedings of a Conference in Honor of A.O.L. Atkin (D. A. Buell and J. T. Teitelbaum, eds.), Studies in Advanced Mathematics, vol. 7, American Mathematical Society, 1998, pp. 21–76.
9. Andreas Enge, `mpfrcx` — *a library for univariate polynomials over arbitrary precision real or complex numbers*, Version 0.1, `http://www.lix.polytechnique.fr/Labo/Andreas.Enge/Software.html`.
10. ———, *The complexity of class polynomial computation via floating point approximations*, HAL-INRIA 1040 and ArXiv cs.CC/0601104, INRIA, 2006, `http://hal.inria.fr/inria-00001040`; submitted to *Mathematics of Computation*.
11. Andreas Enge and François Morain, *Comparing invariants for class fields of imaginary quadratic fields*, Algorithmic Number Theory — ANTS-V (Berlin) (Claus Fieker and David R. Kohel, eds.), Lecture Notes in Computer Science, vol. 2369, Springer-Verlag, 2002, pp. 252–266.
12. ———, *SEA in genus 1: 2500 decimal digits*, December 2006, Posting to the Number Theory List, `http://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0612&L=NMBRTHRY&P=R125&I%=-3`.
13. ———, *Further investigations of the generalised Weber functions*, In preparation, 2007.

14. Andreas Enge and Reinhard Schertz, *Constructing elliptic curves over finite fields using double eta-quotients*, Journal de Théorie des Nombres de Bordeaux **16** (2004), 555–568.
15. ______, *Modular curves of composite level*, Acta Arithmetica **118** (2005), no. 2, 129–141.
16. Andreas Enge and Paul Zimmermann, `mpc` — *a library for multiprecision complex arithmetic with exact rounding*, Version 0.4.6, `http://www.lix.polytechnique.fr/Labo/Andreas.Enge/Software.html`.
17. Mireille Fouquet and François Morain, *Isogeny volcanoes and the SEA algorithm*, Algorithmic Number Theory — ANTS-V (Berlin) (Claus Fieker and David R. Kohel, eds.), Lecture Notes in Computer Science, vol. 2369, Springer-Verlag, 2002, pp. 276–291.
18. Martin Fürer, *Faster integer multiplication*, Proceedings of the 39th Annual ACM Symposium on Theory of Computing — STOC'07 (New York) (Association for Computing Machinery, ed.), ACM, 2007, pp. 57–66.
19. Steven D. Galbraith, *Constructing isogenies between elliptic curves over finite fields*, LMS Journal of Computation and Mathematics **2** (1999), no. 2, 118–138.
20. Joachim von zur Gathen and Jürgen Gerhard, *Modern computer algebra*, Cambridge University Press, 1999.
21. Pierrick Gaudry, *Assembly support for GMP on AMD64*, April 2005, `http://www.loria.fr/~gaudry/mpn_AMD64/`.
22. Torbjörn Granlund et al., `gmp` — *GNU multiprecision library*, Version 4.2.1, `http://gmplib.org/`.
23. Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann et al., `mpfr` — *a library for multiple-precision floating-point computations with exact rounding*, Version 2.2.1, `http://www.mpfr.org`.
24. William B. Hart, *Evaluation of the Dedekind eta funktion*, Ph.D. thesis, Macquarie University, 2004.
25. David Kohel, *Endomorphism rings of elliptic curves over finite fields*, Ph.D. thesis, University of California at Berkeley, 1996.
26. François Morain, *Calcul du nombre de points sur une courbe elliptique dans un corps fini: aspects algorithmiques*, Journal de Théorie des Nombres de Bordeaux **7** (1995), no. 1, 111–138.
27. Volker Müller, *Ein Algorithmus zur Bestimmung der Punktanzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer drei*, Dissertation, Universität des Saarlandes, Saarbrücken, 1995.
28. Andrew P. Ogg, *Hyperelliptic modular curves*, Bulletin de la Société Mathématique de France **102** (1974), 449–462.
29. L. Schläfli, *Beweis der Hermiteschen Verwandlungstafeln für die elliptischen Modulfunktionen*, Journal für die reine und angewandte Mathematik **72** (1870), 360–369.
30. A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292.
31. Jacques Vélu, *Isogénies entre courbes elliptiques*, Comptes Rendus de l'Académie des Sciences de Paris, Série A **273** (1971), 238–241.
32. Heinrich Weber, *Lehrbuch der Algebra*, 2nd ed., vol. 3: *Elliptische Funktionen und algebraische Zahlen*, Vieweg, Braunschweig, 1908.

INRIA Saclay–Île-de-France & Laboratoire d'Informatique (CNRS/UMR 7161), École polytechnique, 91128 Palaiseau Cedex, France

E-mail address: `enge@lix.polytechnique.fr`