MATHEMATICS OF COMPUTATION
Volume 87, Number 310, March 2018, Pages 893–904
http://dx.doi.org/10.1090/mcom/3231
Article electronically published on September 7, 2017

ON THE EVALUATION OF SOME SPARSE POLYNOMIALS

DORIAN NOGNENG AND ÉRIC SCHOST

ABSTRACT. We give algorithms for the evaluation of sparse polynomials of the form

$$P = p_0 + p_1 x + p_2 x^4 + \cdots + p_{N-1} x^{(N-1)^2},$$

for various choices of coefficients p_i. First, we take $p_i = p^i$, for some fixed p; in this case, we address the question of fast evaluation at a given point in the base ring, and we obtain a cost quasi-linear in $\sqrt{N}$. We present experimental results that show the good behavior of this algorithm in a floating-point context, for the computation of Jacobi theta functions.

Next, we consider the case of arbitrary coefficients; for this problem, we study the question of multiple evaluation: we show that one can evaluate such a polynomial at N values in the base ring in subquadratic time.

1. INTRODUCTION

The evaluation and interpolation of polynomials, either dense or sparse, are fundamental problems in computer algebra, and have been studied for decades. One particular reason for their importance is that these problems form the basis of a host of *modular algorithms* [13, Chapter 5], one of the key examples being polynomial multiplication by means of Fast Fourier Transform techniques.

Consider the polynomial

$$P = p_0 + p_1 x + \cdots + p_{N-1} x^{N-1},$$

with coefficients in a ring $\mathbb{A}$. Given q in $\mathbb{A}$, for general coefficients $p_0, \ldots, p_{N-1}$, computing $P(q)$ takes $\Theta(N)$ operations in $\mathbb{A}$; for the actual constants involved in the big-Theta, in terms of additions and multiplications, see [24, 25]. Although one cannot improve on such a linear-time bound, it is, however, possible to obtain meaningful results by looking at slight variations of the problem:

- For some choices of coefficients $p_0, \ldots, p_{N-1}$, it is possible to evaluate the corresponding polynomial P in sublinear time: this is the case if the p_i's are polynomial in i (for a polynomial whose degree is assumed to be constant), or of the form c^i for some constant c, ...
- If several evaluations are needed, savings are possible: for instance, evaluation at N points can be done in $O(\mathsf{M}(N)\log(N))$ operations in $\mathbb{A}$, using the divide-and-conquer algorithm described in [13, Chapter 10]. Here, and throughout this paper, $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ is such that one can multiply polynomials of degree N in $\mathbb{A}[x]$ in $\mathsf{M}(N)$ operations; we also ask that the super-linearity conditions of [13, Chapter 8] are satisfied. Since one can take

Received by the editor February 16, 2016 and, in revised form, February 23, 2016 and September 16, 2016.

2010 *Mathematics Subject Classification.* Primary 68W30; Secondary 11Y16.

Key words and phrases. Sparse polynomials, evaluation.

$\mathsf{M}(n)$ in $O(N\log(N)\log\log(N))$ using the Cantor-Kaltofen algorithm [9], this shows that for evaluation at N points, the amortized cost per point is logarithmic in N.

In this paper, we are interested in similar questions for evaluating *sparse* polynomials. Consider a polynomial P of the form

$$P = p_0x^{e_0} + p_1x^{e_1} + \cdots + p_{N-1}x^{e_{N-1}},$$

for some increasing sequence of non-negative integers e_i. For a general choice of coefficients $p_0, \dots, p_{N-1}$, evaluating P at a point requires to read all of these coefficients, so the cost is $\Omega(N)$; on the other hand, using repeated squaring, an obvious upper bound on the cost of computing $P(q)$, for q in $\mathbb{A}$, is $O(N\log(e_{N-1}))$. Provided e_{N-1} is polynomial in N, this is $O(N\log(N))$, so the lower and upper bound differ by at most a logarithmic factor.

In this context, it makes sense to consider again the two questions above, evaluation for special choices of coefficients, and evaluation at multiple points. It would be highly desirable to obtain results for arbitrary choices of exponents; in this paper, we take the first steps in this direction by addressing the case where the exponent e_i takes the form $e_i = i^2$, so that we have

$$P = p_0 + p_1x + p_2x^4 + \cdots + p_{N-1}x^{(N-1)^2}.$$

More generally, we will be able to take $e_i = ai^2 + bi$, for some rationals a, b chosen such that e_i remains a non-negative integer for all i. Explicitly, we call a pair $\boldsymbol{k} = (a, b)$ in $\mathbb{Q}^2$ *admissible* if $ai^2 + bi$ takes non-negative integer values for all i in $\mathbb{N}$; this is equivalent to $a + b$ and $2a$ being themselves non-negative integers.

The questions we consider are then the following:

- Working over a ring $\mathbb{A}$ as before, take an admissible pair $\boldsymbol{k} = (a, b)$ in $\mathbb{Q}^2$ and p in $\mathbb{A}$, and define the polynomial having coefficients $p_i = p^i$ and exponents $e_i = ai^2 + bi$:

  $$P_{N,\boldsymbol{k},p} = \sum_{0\le i<N} p^i x^{ai^2+bi} \in \mathbb{A}[x].$$

  For such a polynomial, we will obtain in section 2 a bound of $O\tilde{~}(\sqrt{N})$ operations in $\mathbb{A}$ for the evaluation at a point $q \in \mathbb{A}$; here, and in the rest of this paper, the $O\tilde{~}$ notation indicates the omission of polylogarithmic factors.
- For the case of general coefficients, we obtain in section 3 a bound of $O\tilde{~}(N^{(\omega+5)/4})$ for multiple evaluation at N points in $\mathbb{A}$. Here, ω is such that over any ring $\mathbb{A}$, one can multiply $r \times r$ matrices in $O(r^\omega)$ operations. Using the latest refinements to date on the Coppersmith-Winograd algorithm [22], one can take $\omega < 2.3728639$; we will assume $\omega > 2$ below, in order to slightly simplify some bounds.

  As soon as $\omega < 3$, the resulting exponent $(\omega + 5)/4$ in less than 2, so the total cost is subquadratic. This is to be compared to the direct algorithm that does N distinct evaluations, with cost $O(N^2\log(N))$ (as we will see in section 3, this can easily be improved to $O(N^2)$).

The polynomials $P_{N,\boldsymbol{k},p}$ with coefficients in geometric progression seen in our particular case are obtained by truncating the sums defining Jacobi theta functions such as

$$\vartheta(z;\tau) = \sum_{n=-\infty}^{\infty} e^{\pi i(2nz+n^2\tau)} = \sum_{n=-\infty}^{\infty} \eta^n q^{n^2}, \tag{1}$$

with $\eta = e^{2i\pi z}$ and $q = e^{i\pi\tau}$ (the above is often written ϑ_3). These functions show up in a number of situations, often of a geometric or number theoretic nature; see, for instance, [23] in a context of algebraic geometry (construction of Abelian varieties), [7, Chapters 2, 3, 9] for relations to the Arithmetic Geometric Mean and elliptic integrals or [4] for connections to the heat equation. The Dedekind eta function, which is the specialization of a theta function at suitable choices of the arguments, also admits such an expansion:

$$\eta(\tau) = e^{i\pi\tau/12} \sum_{n=-\infty}^{\infty} (-1)^n q^{n(3n-1)/2}, \quad \text{with this time } q = e^{2i\pi\tau}.$$

Sums such as $\sum_{0\le i<N} x^{ai^2+bi}$, that is, $P_{N,\boldsymbol{k},1}$ for our notation above, also show up in the algorithm of [26] for the construction of prime numbers. Lemma 3.1 in that reference is the only previous result known to us for the evaluation of polynomials $P_{N,\boldsymbol{k},p}$ at an arbitrary point with sublinear complexity; the cost given in the proof of the lemma is $O(N^{\omega/3})$, so our cost of $O\tilde{~}(\sqrt{N})$ is an improvement on that result.

Particular cases include as well polynomials $\sum_{0\le i<N} x^{i^2}$, whose values at points of the form $e^{2i\pi s/N}$ are known as (quadratic) Gauss sums and are used to prove, e.g., quadratic reciprocity [1].

Because of their importance in analytic contexts, obtaining fast algorithms that approximate truncated sums as above is highly relevant, see for instance applications in [11] to the computation of modular polynomials or in [16, 17] to the computation of the Riemann zeta function. In this context, we mention the results of Labrande [21], building on previous work by Dupont [10], which give an algorithm for the computation of theta functions with quasi-linear bit complexity in N, the requested precision.

One should be careful when trying to extend to such floating point calculation results of an algebraic complexity nature such as the ones we obtain in this paper. In order to get an understanding of the behavior of our algorithm in this context, we implemented the use of the Arb library [18] (a C library for arbitrary-precision floating-point ball arithmetic). In section 2, we present experimental results for the computation of theta functions; for high precision, our algorithm improves on the algorithm built-in in Arb, and on Labrande's implementation of his algorithm.

Regarding the algorithm in section 3, we are not aware of previous results on the cost of the multiple evaluation of such polynomials at general points. An algorithm due to Canny, Kaltofen and Lakshman [8] gives a quasi-linear time algorithm for the evaluation at points in *geometric progression* of an arbitrary sparse polynomial, using the fact that the matrix of the resulting linear map is transpose-Vandermonde. There also exists extensive literature on the *interpolation* of sparse polynomials, of which [2, 3, 5, 12, 14, 19, 20, 27] is only a sample. For such interpolation algorithms, determining the exponents is usually the main issue; once they are known, choosing again interpolation points in geometric progression makes it rather straightforward to recover the coefficients.

2. Coefficients in geometric progression

2.1. **Main results.** Consider an admissible pair $\boldsymbol{k} = (a, b)$, an element p in a ring $\mathbb{A}$, and recall the definition of the polynomial $P_{N,\boldsymbol{k},p}$:

$$P_{N,\boldsymbol{k},p} = \sum_{0 \le i < N} p^i x^{ai^2+bi} \in \mathbb{A}[x].$$

We keep N and $\boldsymbol{k}$ as indices in such a notation, as we will have to change their values later on, and wish to avoid all ambiguities. Our goal is to compute $P_{N,\boldsymbol{k},p}(q)$, for a given q in $\mathbb{A}$. As pointed out in the introduction, this can be done in a naive manner using $O(N \log(N))$ operations; the following proposition gives an upper bound softly linear in $\sqrt{N}$.

Unless otherwise specified, we assume in our cost estimates that the entries a and b in $\boldsymbol{k}$ are fixed constants.

Proposition 1. *There exist algorithms* $\mathsf{Eval}_{\mathrm{ff}}$ *(fraction free) and* $\mathsf{Eval}_{\mathrm{inv}}$ *(relying on inversion) with the following characteristics:*

- *on input q in $\mathbb{A}$,* $\mathsf{Eval}_{\mathrm{ff}}$ *computes $P_{N,\boldsymbol{k},p}(q)$ using $O(\mathsf{M}(\sqrt{N}) \log(N))$ operations $+, -, \times$ in $\mathbb{A}$;*
- *on input q in $\mathbb{A}^*$ and $1/q$,* $\mathsf{Eval}_{\mathrm{inv}}$ *computes $P_{N,\boldsymbol{k},p}(q)$ using $O(\mathsf{M}(\sqrt{N}))$ operations $+, -, \times$ in $\mathbb{A}$.*

The only difference between these two algorithms lies in a subroutine for multipoint evaluation of a polynomial. Given a ring $\mathbb{A}$, a polynomial F in $\mathbb{A}[x]$ of degree less than some integer N, and an element q in $\mathbb{A}$, consider the question of computing the values $F(q^{2i})$ for $i = 0, \dots, N-1$. This can be done using $O(\mathsf{M}(N) \log(N))$ operations $+, -, \times$ in $\mathbb{A}$ using the divide-and-conquer algorithm of [13, Chapter 10] that was already mentioned in the introduction. However, if q is a unit and $1/q$ is known, the Bluestein algorithm [6] reduces the cost to $\mathsf{M}(N) + O(N)$ operations $+, -, \times$.

Since we assume that a and b are constant, the cost given in the proposition above only depends on N, so there is no possible confusion as to the meaning of the big-O notation. Below, we also write big-O's involving several variables; in that case, $f(a, b, \dots) = O(g(a, b, \dots))$, for non-negative integers $a, b, \dots$, will mean that:

- f and g are defined at all values of the variables, except maybe finitely many;
- there exists K in $\mathbb{R}$ such that for all values of the variables, except maybe finitely many, $f(a, b, \dots) \le K g(a, b, \dots)$.

In particular, we will avoid using the function log in this context (since it may violate the first condition); instead, we will use the function $\mathrm{logp} : \mathbb{N} \to \mathbb{R}$, defined as $\mathrm{logp}(a) = \log(\max(a, 1))$, so that $\mathrm{logp}(0) = 0$.

2.2. **A direct algorithm.** In this subsection, we consider an admissible pair $\boldsymbol{\kappa} = (\alpha, \beta)$ in $\mathbb{Q}^2$, together with an element p in $\mathbb{A}$. For integers μ, ν, with $\mu \le \nu$, define the polynomial

$$P_{\mu,\nu,\boldsymbol{\kappa},p} = \sum_{\mu \le i < \nu} p^i x^{\alpha i^2 + \beta i}.$$

There are at most ν terms in this sum; if we assume that $\boldsymbol{\kappa} = (\alpha, \beta)$ is constant, each of them can be computed using $O(\log(\nu))$ multiplications using repeated squaring.

Hence, as was already mentioned in the introduction for $\mu = 0$, we can compute $P_{\mu,\nu,\boldsymbol{\kappa},p}(q)$ using $O(\nu \log(\nu))$ operations $+, \times$ in $\mathbb{A}$.

It is folklore that one can do slightly better, using suitable addition chains; this is the object of the following lemma and its corollary. In this lemma, *we do not consider that $\boldsymbol{\kappa}$ is constant*, so we give costs that involve (α, β). Since by convention our multi-variable big-Os involve only non-negative integer arguments, we introduce $u = 2\alpha$ and $v = \alpha + \beta$; both lie in $\mathbb{N}$, by assumption on (α, β).

Lemma 1. *For μ, ν and $\boldsymbol{\kappa}$ as above, given a ring $\mathbb{A}$ and p, q in $\mathbb{A}$, one can compute $(p^i q^{\alpha i^2+\beta i})_{\mu \le i < \nu}$ using*

$$O(\mathrm{logp}(u+v) + \mathrm{logp}(\mu) + (\nu - \mu))$$

multiplications in $\mathbb{A}$.

Note that under our convention, the bound above is $O(\mathrm{logp}(u+v) + \mathrm{logp}(\nu) + (\nu - \mu))$, but *not* $O(\mathrm{logp}(u+v) + (\nu - \mu))$, as can be seen by looking at the situation $u = v = 0$ and $\mu = (\nu - 1) \to \infty$.

Proof. Define the sequences $(r_i)_{i \in \mathbb{N}}$ and $(s_i)_{i \in \mathbb{N}}$ by $r_i = pq^{2\alpha i + \alpha + \beta}$ and $s_i = p^i q^{\alpha i^2 + \beta i}$, so that we have for i in $\mathbb{N}$,

$$r_{i+1} = q^{2\alpha} r_i,$$
$$s_{i+1} = r_i s_i.$$

Given p and q in $\mathbb{A}$, this will allow us to compute all $(s_i)_{\mu \le i < \nu} = (p^i q^{\alpha i^2+\beta i})_{\mu \le i < \nu}$ in the claimed cost. As mentioned above, we will write $u = 2\alpha$ and $v = \alpha + \beta$ below.

Computing $q^{2\alpha} = q^u$ can be done by repeated squaring using $O(\mathrm{logp}(u)) = O(\mathrm{logp}(u+v))$ multiplications. Similarly, the values of

$$(r_\mu, s_\mu) = (pq^{u\mu+v}, p^\mu q^{u\mu(\mu-1)/2+v\mu})$$

can be obtained using

$$O(1 + \mathrm{logp}(\mu) + \mathrm{logp}(u\mu + v) + \mathrm{logp}(u\mu(\mu-1)/2 + v\mu))$$

multiplications, which is seen to be

$$O(\mathrm{logp}(u+v) + \mathrm{logp}(\mu)).$$

Then, the formula above shows that we can deduce the values of (r_{i+1}, s_{i+1}) from those of (r_i, s_i) using $O(1)$ operations. As a result, the sequence $(r_\mu, s_\mu), \ldots, (r_{\nu-1}, s_{\nu-1})$ can be deduced in $O(\nu - \mu)$ operations. Summing all costs proves the lemma. □

In the particular case where the entries of $\boldsymbol{\kappa}$ are fixed constants, we deduce the following consequence.

Corollary 1. *For μ, ν and $\boldsymbol{\kappa}$ as above, with $\boldsymbol{\kappa}$ constant, given a ring $\mathbb{A}$ and p, q in $\mathbb{A}$, one can compute $P_{\mu,\nu,\boldsymbol{\kappa},p}(q)$ using $O(\mathrm{logp}(\mu) + (\nu - \mu))$ multiplications in $\mathbb{A}$.*

Proof. It suffices to compute all elements $(p^i q^{\alpha i^2+\beta})_{\mu \le i < \nu}$ and add them. Since $\boldsymbol{\kappa} = (\alpha, \beta)$, and thus u and v, is supposed to be constant, the only terms remaining in the cost estimate of the previous lemma are $O(\mathrm{logp}(\mu) + (\nu - \mu))$. □

2.3. **The main algorithm.** Let us consider again the question of computing $P_{N,\boldsymbol{k},p}(q)$, for some given admissible pair $\boldsymbol{k} = (a, b)$. We assume here that all entries in $\boldsymbol{k}$ are fixed constants.

We first reduce to the case where N is a square. Let $m_0 = \lfloor\sqrt{N}\rfloor$; if m_0 is even, we take $m = m_0$, otherwise we take $m = m_0 - 1$, so that in any case m is even. Define $M = m^2$; thus, we have

$$P_{N,\boldsymbol{k},p} = P_{M,\boldsymbol{k},p} + \sum_{M\le i<N} p^i x^{ai^2+bi} = P_{M,\boldsymbol{k},p} + P_{M,N,\boldsymbol{k},p},$$

using the notation of the previous subsection. The difference $N - M$ is $O(\sqrt{N})$, so by Corollary 1, given q in $\mathbb{A}$ we can compute $P_{M,N,\boldsymbol{k},p}(q)$ using $O(\sqrt{N})$ operations. We will now focus on the calculation of $P_{M,\boldsymbol{k},p}(q)$, which will take a comparable cost; the cost given in the following lemma, together with the estimate above for $P_{M,N,\boldsymbol{k},p}$, is then enough to prove Proposition 1.

Lemma 2. *Suppose that $\boldsymbol{k}$ is fixed. There exist algorithms* $\mathsf{EvalSquare}_{\mathrm{ff}}$ *(fraction free) and* $\mathsf{EvalSquare}_{\mathrm{inv}}$ *(relying on inversion) with the following characteristics: for any ring $\mathbb{A}$, given $M = m^2$ as above,*

- *on input q in $\mathbb{A}$,* $\mathsf{EvalSquare}_{\mathrm{ff}}$ *computes $P_{M,\boldsymbol{k},p}(q)$ using $O(\mathsf{M}(\sqrt{M})\log(N))$ operations $+, -, \times$ in $\mathbb{A}$;*
- *on input q in $\mathbb{A}^*$ and $1/q$,* $\mathsf{EvalSquare}_{\mathrm{inv}}$ *computes $P_{M,\boldsymbol{k},p}(q)$ using $O(\mathsf{M}(\sqrt{N}))$ operations $+, -, \times$ in $\mathbb{A}$.*

Proof. Decomposing $i \in \{0, \dots, M - 1\}$ as $i = j + mk$, with both j and k in $\{0, \dots, m - 1\}$, we can rewrite $P_{M,\boldsymbol{k},p}$ as

$$\begin{aligned} P_{M,\boldsymbol{k},p} &= \sum_{0\le i<M} p^i x^{ai^2+bi} \\ &= \sum_{0\le j,k<m} p^{(j+mk)} x^{a(j+mk)^2+b(j+mk)} \\ &= \sum_{0\le j,k<m} p^{j+mk} x^{aj^2+bj+2amjk+am^2k^2+bmk}. \end{aligned}$$

It follows that if we define

$$L_{M,\boldsymbol{k},p} = \sum_{0\le j<m} p^j x^{aj^2+bj} y^j \in \mathbb{A}[x, y],$$

we have

$$P_{M,\boldsymbol{k},p} = \sum_{0\le k<m} L_{M,\boldsymbol{k},p}(x, x^{2amk}) p^{mk} x^{am^2k^2+bmk}.$$

This leads us to the following algorithm:

(1) Compute all powers

$$s_j = p^j q^{aj^2+bj} \quad \text{and} \quad t_k = p^{mk} q^{am^2k^2+bmk} \quad \text{for } j \text{ and } k \text{ in } \{0, \dots, m-1\}.$$

The left-hand terms s_j come from a direct application of Lemma 1, with input parameters $\boldsymbol{\kappa} = (a, b)$, p, q and $(\mu, \nu) = (0, m)$; since (a, b) are assumed to be constant, the cost is $O(m)$ operations.

For the right-hand terms t_k, we first compute p^m using $O(\log(m))$ operations. Then, we recall Lemma 1 again, with input parameters $\boldsymbol{\kappa} = (am^2, bm)$, p^m, q and $(\mu, \nu) = (0, m)$, and thus with $u + v = O(m^2)$. The cost is now $O(\log(m^2) + m)$, which remains $O(m)$.

(2) Compute $\xi = q^{am}$; because m is even, the exponent is an integer, so this is well-defined. The cost is $O(\log(m))$ operations.

(3) Form the polynomial $\lambda = L_{M,\boldsymbol{k},p}(q, y) = \sum_{0 \le j < m} s_j y^j$ in $\mathbb{A}[y]$, and compute the values

$$v_k = \lambda(\xi^{2k}) \quad \text{for } 0 \le k < m.$$

In view of the discussion in subsection 2.1, this can be done in either $O(\mathsf{M}(m)\log(m))$ or $O(\mathsf{M}(m))$ operations in $\mathbb{A}$, depending on whether q is a unit, and its inverse is known, or not.

(4) Compute and return $\sum_{0 \le k < m} v_k t_k$, for a cost of $O(m)$.

The sum of all costs seen so far is either $O(\mathsf{M}(m)\log(m))$ or $O(\mathsf{M}(m))$, depending on our choice of the algorithm for evaluating λ, so the lemma is proved. □

2.4. **Experimental results.** We implemented the previous algorithm in two different contexts: over a small prime field using the C library FLINT [15], and using ball arithmetic using the Arb library as an extension of FLINT [18]. All timings are obtained on an Intel i7-5600U, with 8GB RAM, and given in seconds.

Figure 1 shows results obtained when computing an N term sum of the kind seen above (for N a square); we compare a direct algorithm using a naive addition chain to the algorithm $\mathsf{Eval}_{\mathrm{inv}}$ of Proposition 1. For such computations, arithmetic operations count as a rather reliable prediction of practical performance; the new algorithm is significantly faster than the addition chain.

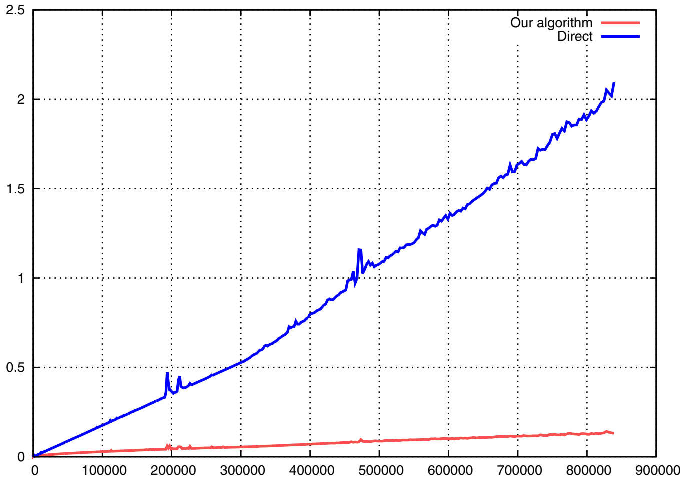

FIGURE 1. Experiments modulo 1125899906842679; number of terms vs. time in seconds.

Figure 2 gives results for computing approximations of the theta function $\vartheta(z;\tau)$ given in equation (1). Given a target precision ε, we determine N such that the truncated sums

$$\sum_{n=0}^{N-1} \eta^n q^{n^2} + \sum_{n=0}^{N-1} \eta^{-n} q^{n^2} - 1$$

approximate $\vartheta(z;\tau)$ with error at most ε, with $\eta = e^{2i\pi z}$ and $q = e^{i\pi\tau}$. Following [21], in our tests, we take $z = 0.123456789 + 0.123456789i$ and $\tau = 0.23456789 + 1.23456789i$, which ensures sufficiently rapid convergence to obtain meaningful intervals in the output.

We compare our implementation to the built-in Arb routine `acb_modular_theta_sum`, which uses an addition chain (denoted by Arb in Figure 2), and with Labrande's implementation of his algorithm [21] (denoted by FastThetas in Figure 2). The library Arb performs all operations using ball arithmetic, so that an error bound is automatically attached to each variable, allowing us to assess the quality of the output; Labrande's algorithm involves an iterative process, whose proof relies on a detailed precision analysis. It is important to note that these previous algorithms compute several theta functions at once; if only one is needed, as here, savings may be possible, albeit by an amount that is unclear to us. In our experiments, the arguments are fixed to the values given above, and the requested precision varies as $\varepsilon = 2^{-p}$ (p is in abscissa in Figure 2).

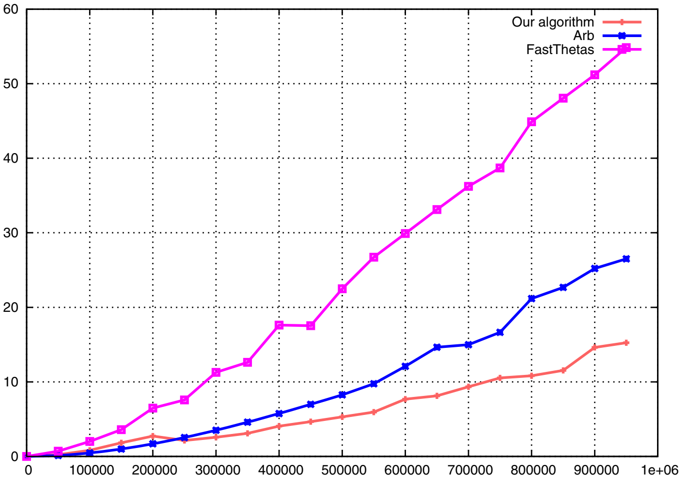

FIGURE 2. Experiments using high precision arithmetic; precision vs. time in seconds.

In this context, for our algorithm, calculations are done using ball arithmetic; as a result, the cost analysis of Proposition 1 is not expected to be a good model of practical performance, since it overlooks all issues related to size of coefficients and precision management. At the heart of our algorithm, Bluestein's evaluation algorithm involves a product of polynomials, one of them having rapidly decreasing coefficients of the form q^{i^2}, $i = 0, \ldots$. This may possibly make it poorly suited to fast computations in a ball arithmetic context, and as a result, it is hard to predict whether our approach will turn out to be superior to others.

Our experiments show that this is partially the case, as our algorithm outperforms the other implementations for high precisions. A more precise analysis is required to understand this behavior; this would necessarily have to take into account the specifics of the implementation of polynomial arithmetic in Arb. In terms of quality of the output, the differences between the three outputs were always significantly smaller than the requested precision 2^{-p}.

3. Arbitrary coefficients

In this section, we consider the case of a polynomial with arbitrary coefficients, and exponents as before, that is,

$$P = \sum_{0 \le i < N} p_i \, x^{ai^2+bi}.$$

Since both the number of terms N and the exponents $ai^2 + bi$ are fixed throughout this section, there is no need to make them explicit as indices, as we did in the previous section.

As explained in the introduction, evaluating P at a given point $q \in \mathbb{A}$ takes time $\Omega(N)$, since we have to use all coefficients. This bound is sharp, since using the same techniques as in Corollary 1, we can compute $P(q)$ in linear time. In this section, we show that we can do better when several evaluations are needed.

Proposition 2. *The following hold:*

- *for any monic polynomial M of degree N in $\mathbb{A}[X]$, one can compute P mod M using $O(N^{(\omega+2)/4}\mathsf{M}(N^{3/4})) = \tilde{O}(N^{(\omega+5)/4})$ operations in $\mathbb{A}$;*
- *given $q_0, \dots, q_{N-1}$ in $\mathbb{A}$, one can compute $(P(q_i))_{0 \le i < N}$ using $O(N^{(\omega+2)/4}\mathsf{M}(N^{3/4})) = \tilde{O}(N^{(\omega+5)/4})$ operations in $\mathbb{A}$.*

The second part follows readily from the first one: given $q_0, \dots, q_{N-1}$, we can construct the polynomial $M = (x - q_0) \cdots (x - q_{N-1})$ using $O(\mathsf{M}(N)\log(N))$ operations in $\mathbb{A}$ using the algorithm of [13, Chapter 10]. Applying the first part of the proposition, we compute $Q = P \bmod M$ in $O(N^{(\omega+2)/4}\mathsf{M}(N^{3/4}))$ operations in $\mathbb{A}$. Finally, we evaluate Q at $q_0, \dots, q_{N-1}$, using the multi-point evaluation algorithm of [13, Chapter 10]; the cost is again $O(\mathsf{M}(N)\log(N))$. The dominant cost is the computation of P mod M, so we can focus on proving the first assertion.

Without loss of generality, by adding $O(N)$ zero coefficients, we may suppose that N is a fourth power, that is, $N = n^4$ for some integer n. This allows us to write any index i in $\{0, \dots, N-1\}$ uniquely under the form $i = j + kn + \ell n^2$, with j, k in $\{0, \dots, n-1\}$ and ℓ in $\{0, \dots, n^2 - 1\}$, which leads to the decomposition

$$\begin{aligned} P &= \sum_{0 \le i < N} p_i \, x^{ai^2+bi} \\ &= \sum_{0 \le \ell < n^2} \sum_{0 \le j,k < n} p_{j+kn+\ell n^2} \, x^{a(j+kn+\ell n^2)^2 + b(j+kn+\ell n^2)} \\ &= \sum_{0 \le \ell < n^2} \sum_{0 \le j,k < n} p_{j+kn+\ell n^2} \, x^{a(j^2+k^2n^2+\ell^2n^4+2jkn+2j\ell n^2+2k\ell n^3)+b(j+kn+\ell n^2)} \\ &= \sum_{0 \le \ell < n^2} \sum_{0 \le k < n} x^{a(k^2n^2+\ell^2n^4+2k\ell n^3)+b(kn+\ell n^2)} \\ &\quad \times \sum_{0 \le j < n} p_{j+kn+\ell n^2} \, x^{a(j^2+2jkn+2j\ell n^2)+bj}. \end{aligned}$$

For indices j, k, ℓ as above, define the polynomials

$$\begin{aligned} A_{k,\ell} &= x^{a(k^2n^2+\ell^2n^4+2k\ell n^3)+b(kn+\ell n^2)} \mod M, \\ B_{j,\ell} &= x^{a(j^2+2j\ell n^2)+bj} \mod M; \end{aligned}$$

these definitions make sense, since

$$a(k^2n^2 + 2k\ell n^3 + \ell^2 n^4) + b(kn + \ell n^2) = a(kn + \ell n^2)^2 + b(kn + \ell n^2)$$

is a non-negative integer, and similarly for

$$a(j^2 + 2j\ell n^2) + bj = (aj^2 + bj) + (2a)j\ell n^2.$$

This yields

$$P \mod M = \sum_{0\le \ell<n^2} \sum_{0\le k<n} A_{k,\ell} \sum_{0\le j<n} p_{j+kn+\ell n^2}\, x^{2ajkn} B_{j,\ell} \mod M.$$

The point behind this particular decomposition is that the terms x^{2ajkn} all have small degree; we use this remark in the following lemma. First, however, notice that there are $O(n^3)$ polynomials $A_{k,\ell}$ and $B_{j,\ell}$. Since the exponents involved are all $O(N^2) = O(n^8)$, each such polynomial can be computed by repeatedly squaring modulo M, using $O(\mathsf{M}(n^4)\log(n))$ operations in $\mathbb{A}$, for a total of $O(n^3\mathsf{M}(n^4)\log(n))$ operations.

Lemma 3. *For any ℓ in $\{0, \dots, n^2-1\}$, one can compute all*

$$C_{k,\ell} = \sum_{0\le j<n} p_{j+kn+\ell n^2}\, x^{2ajkn} B_{j,\ell} \mod M$$

for k in $\{0, \dots, n-1\}$, using $O(n^\omega \mathsf{M}(n^3))$ operations in $\mathbb{A}$.

Proof. For given k and ℓ, computing a single polynomial $C_{k,\ell}$ amounts to doing the dot-product between the polynomial vectors $(p_{j+kn+\ell n^2}\, x^{2ajkn})_{0\le j<n}$ (seen as a row) and $(B_{j,\ell})^t_{0\le j<n}$ (seen as a column). Because we assume that a is constant, the monomial x^{2ajkn} has degree $O(n^3)$, whereas $B_{j,\ell} \mod M$ has degree less than n^4. Let us write

$$B_{j,\ell} = \sum_{0\le s<n} B_{j,\ell,s} x^{sn^3},$$

with all $B_{j,\ell,s}$ of degree less than n^3; computing this decomposition is free, as it amounts to coefficient extraction.

Let us then compute the row vector $(C_{k,\ell,s})_{0\le s<n}$ obtained as the product of the row vector $(p_{j+kn+\ell n^2}\, x^{2ajkn})_{0\le j<n}$ and the matrix $(B_{j,\ell,s})_{0\le j,s<n}$. Given this vector, one can recover $C_{k,\ell} = \sum_{0\le s<n} C_{k,\ell,s} x^{sn^3}$ in linear time $O(n^4)$, since all $C_{k,\ell,s}$ have degree $O(n^3)$.

Let us now fix ℓ, and vary k. To compute all $C_{k,\ell}$, for k in $\{0, \dots, n-1\}$, we are led to multiply the polynomial matrices $(p_{j+kn+\ell n^2}\, x^{2ajkn})_{0\le k,j<n}$ and $(B_{j,\ell,s})_{0\le j,s<n}$. Since all polynomials involved have degree $O(n^3)$, this product can be obtained in $O(n^\omega \mathsf{M}(n^3))$ operations in $\mathbb{A}$. As a consequence of what we saw above, recovering all $C_{k,\ell}$ takes negligible time $O(n^5)$. □

With this lemma, we can conclude the proof of Proposition 2. The lemma shows that all polynomials $C_{k,\ell}$ can be computed using $O(n^{\omega+2}\mathsf{M}(n^3))$ operations in $\mathbb{A}$.

From this, we can recover $P \bmod M$ as

$$P \bmod M = \sum_{0 \le \ell < n^2} \sum_{0 \le k < n} A_{k,\ell}\, C_{k,\ell} \bmod M.$$

There are n^3 terms in the sum, and each can be computed in $O(\mathsf{M}(n^4))$ operations, so the total time to deduce $P \bmod M$ from all $C_{k,\ell}$ is $O(n^3\mathsf{M}(n^4))$.

Overall, since we assumed that $\omega > 2$, the dominant cost is the one of the matrix products of Lemma 3, which is $O(n^{\omega+2}\mathsf{M}(n^3))$. Since $N = n^4$, this proves Proposition 2.

Acknowledgements

The authors thank Andreas Enge and Fredrik Johansson for helpful discussions, and the referees for their remarks on the first version of this paper. The authors were supported by NSERC and the Canada Research Chairs program.

References

[1] T. M. Apostol, *Introduction to Analytic Number Theory*, Undergraduate Texts in Mathematics, Springer-Verlag, New York-Heidelberg, 1976. MR0434929

[2] A. Arnold, M. Giesbrecht, and D. S. Roche, *Faster sparse interpolation of straight-line programs*, CASC 2013, Springer, 2013, pp. 61–74.

[3] A. Arnold, M. Giesbrecht, and D. S. Roche, *Sparse interpolation over finite fields via low-order roots of unity*, ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2014, pp. 27–34, DOI 10.1145/2608628.2608671. MR3239905

[4] R. Bellman, *A Brief Introduction to Theta Functions*, Athena Series: Selected Topics in Mathematics, Holt, Rinehart and Winston, New York, 1961. MR0125252

[5] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, STOC'88, ACM, 1988, pp. 301–309.

[6] L. I. Bluestein, *A linear filtering approach to the computation of the discrete Fourier transform*, IEEE Transactions on Audio and Electroacoustics **AU-18** (1970), 451–455.

[7] J. M. Borwein and P. B. Borwein, *Pi and the AGM*, Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley & Sons, Inc., New York, 1987. MR877728

[8] J. Canny, E. Kaltofen, and Y. Lakshman, *Solving systems of non-linear polynomial equations faster*, ISSAC'89, ACM, 1989, pp. 121–128.

[9] D. G. Cantor and E. Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta Inform. **28** (1991), no. 7, 693–701, DOI 10.1007/BF01178683. MR1129288

[10] R. Dupont, *Fast evaluation of modular functions using Newton iterations and the AGM*, Math. Comp. **80** (2011), no. 275, 1823–1847, DOI 10.1090/S0025-5718-2011-01880-6. MR2785482

[11] A. Enge, *The complexity of class polynomial computation via floating point approximations*, Math. Comp. **78** (2009), no. 266, 1089–1107, DOI 10.1090/S0025-5718-08-02200-X. MR2476572

[12] S. Garg and É. Schost, *Interpolation of polynomials given by straight-line programs*, Theoret. Comput. Sci. **410** (2009), no. 27-29, 2659–2662, DOI 10.1016/j.tcs.2009.03.030. MR2531108

[13] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd ed., Cambridge University Press, Cambridge, 2003. MR2001757

[14] M. Giesbrecht and D. S. Roche, *Diversification improves interpolation*, ISSAC 2011—Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2011, pp. 123–130, DOI 10.1145/1993886.1993909. MR2895203

[15] W. B. Hart, *Fast library for number theory: An introduction*, ICMS'10 (Berlin, Heidelberg), Springer-Verlag, 2010, `http://flintlib.org`, pp. 88–91.

[16] G. A. Hiary, *Fast methods to compute the Riemann zeta function*, Ann. of Math. (2) **174** (2011), no. 2, 891–946, DOI 10.4007/annals.2011.174.2.4. MR2831110

[17] G. A. Hiary, *A nearly-optimal method to compute the truncated theta function, its derivatives, and integrals*, Ann. of Math. (2) **174** (2011), no. 2, 859–889, DOI 10.4007/annals.2011.174.2.3. MR2831109

[18] F. Johansson, *Arb: a C library for ball arithmetic*, ACM Communications in Computer Algebra **47** (2013), no. 4, 166–169.

[19] E. Kaltofen and L. Yagati, *Improved sparse multivariate polynomial interpolation algorithms*, Symbolic and algebraic computation (Rome, 1988), Lecture Notes in Comput. Sci., vol. 358, Springer, Berlin, 1989, pp. 467–474, DOI 10.1007/3-540-51084-2_44. MR1034749

[20] E. Kaltofen and W.-s. Lee, *Early termination in sparse interpolation algorithms*, J. Symbolic Comput. **36** (2003), no. 3-4, 365–400, DOI 10.1016/S0747-7171(03)00088-9. MR2004034

[21] H. Labrande, *Computing Jacobi's θ in quasi-linear time*, https://hal.inria.fr/hal-01227699, 2015.

[22] F. Le Gall, *Powers of tensors and fast matrix multiplication*, ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2014, pp. 296–303, DOI 10.1145/2608628.2608664. MR3239939

[23] D. Mumford, *Tata Lectures on Theta, 1*, Modern Birkhauser Classics, Springer, 2007.

[24] A. Ostrowski, *On two problems in abstract algebra connected with Horner's rule*, Studies in mathematics and mechanics presented to Richard von Mises, Academic Press Inc., New York, 1954, pp. 40–48. MR0066030

[25] V. Ja. Pan, *On means of calculating values of polynomials* (Russian), Uspehi Mat. Nauk **21** (1966), no. 1 (127), 103–134. MR0207178

[26] T. Tao, E. Croot III, and H. Helfgott, *Deterministic methods to find primes*, Math. Comp. **81** (2012), no. 278, 1233–1246, DOI 10.1090/S0025-5718-2011-02542-1. MR2869058

[27] R. Zippel, *Interpolating polynomials from their values*, J. Symbolic Comput. **9** (1990), no. 3, 375–403, DOI 10.1016/S0747-7171(08)80018-1. MR1056633

LIX, Bâtiment Alan Turing, Campus de l'École Polytechnique, 91120 Palaiseau, France

E-mail address: dorian.nogneng@lix.polytechnique.fr

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

E-mail address: eschost@uwaterloo.ca