2002

TR-2002019: Improved Algorithms for Computing Determinants and Resultants

Ioannis Z. Emiris

Victor Y. Pan

Improved algorithms for computing determinants and resultants

Ioannis Z. Emiris * Victor Y. Pan †

December 10, 2002

Abstract Our first contribution is a substantial acceleration of randomized computation of scalar, univariate, and multivariate matrix determinants, in terms of the output-sensitive bit complexity bounds, including computation modulo a product of random primes from a fixed range. This acceleration is particularly dramatic in a critical application, namely solving polynomial systems and related studies, via computing the resultant. This is achieved by combining our techniques with the primitive-element method, which leads to an effective implicit representation of the roots. We systematically examine quotient formulae of Sylvester-type resultant matrices, including matrix polynomials and the u-resultant. We reduce the known bit complexity bounds by almost an order of magnitude, in terms of the resultant matrix dimension. Our theoretical and practical improvements cover the highly important cases of sparse and degenerate systems.

Keywords computer algebra, randomized algorithms, matrix determinant, bit complexity, structured matrix, polynomial system solving.

1 Introduction

1.1 Our subjects and techniques

The classical problem of computing matrix determinants witnessed dramatic recent progress [EGV00, KV01, Sto02]. Our first step is to adapt the algorithm of [KV01] to the case of modular computation, where the input is a generally multivariate matrix. This shall be an important block for our algorithms, based on Chinese remaindering. We also contribute with a new randomized *output-sensitive* algorithm, which improves the known bounds when the output is away from the available upper bounds. Such upper bounds (like Hadamard's) are notorious for being excessively high in general, as discussed later. These results are of independent interest, but become more important because of their application to polynomial system solving by the method of resultants and their matrices. Our focus on the block Wiedemann approach of [KV01] is motivated by this application where it is currently superior over the others in [EGV00, Sto02].

Computing the resultant R of a polynomial system, including the cases of scalar R and univariate or multivariate polynomial $R(x)$ or $R(u_1, \ldots, u_n)$ is a central practical and theoretical problem in the field of symbolic computation, leading to efficient methods for solving 0-dimensional systems, quantifier elimination, and deciding the theory of the reals. Our practical motivation is the real-time solution of systems in CAD, vision or robotics (which may give rise to matrices with dimension in the hundreds or even higher). Equally useful is the computation of the resultant polynomial, e.g., in modeling applications where an implicit representation of a curve or surface is obtained from the (perturbed) resultant, even in the presence of base points [DE01, MC93]. Resultant values and signs also capture important tests in computational geometry, including the case of infinitesimal perturbations of the input. These diverse applications are discussed in [BEPP99, Can88, CLO98, DE01, Man93].

A polynomial system has a solution iff $R = 0$, whereas computing isolated roots of the system can be reduced to factoring the polynomial $R(u_1, \ldots, u_n)$ into the product of linear factors. Moreover, solving the univariate polynomial equation $R(x) = 0$ projects all common isolated roots to one of their coordinates and is critical in several applications. Thus we arrive at our task of computing the resultants $R, R(x)$, and $R(u_1, \ldots, u_n)$, and in our approach we largely unify this computation. For computing the resultant, we exploit a recent major advance in the field, which generalizes the Macaulay formula to the toric case by expressing the (toric) resultant by a quotient formula $\det M / \det S$, where

*Department of Informatics & Telecommunications, University of Athens, 15771 Greece, and INRIA Sophia-Antipolis, France, emiris@di.uoa.gr. Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

†Mathematics and Computer Science Department, CUNY, Bronx, NY 10566, USA, vpan@lehman.cuny.edu. Partially supported by NSF Grant CCR 9732206 and PSC CUNY Awards 61393-0030, 62435-0031, and 66383-0032.

M is a Sylvester-type resultant matrix of a polynomial system, and S is one of its submatrices [D'A02]. We also apply our methods to the perturbations for handling degenerate systems [Can90, DE01], as well as to Bézout-type matrices.

We focus on *bit complexity*; the computational model is the (Boolean, or logarithmic-cost) *Random Access Machine*, unless otherwise stated [AHU74, BCS97]. Section 7 and the Appendix examine briefly straight-line programs and linear-complexity models, which we apply to capture a specific notion of sparseness. Our estimates do not include the cost of generating the random quantities required in the algorithms, but this complexity is practically dominated at other stages. Randomized algorithms shall be of two types, namely *Las Vegas* or *Monte Carlo* depending on whether the estimated computational cost covers or not the cost of certifying correctness of the output. In the latter case, randomization may lead to a superset of the roots; in system solving applications, certification is very fast since it amounts to evaluating the input polynomials at the computed root values.

Since we deal with highly important and long and well studied problems, our progress had to employ and combine various known advanced techniques of algebraic computing as well as our novel techniques. In particular the latter include multivariate output sensitive version of the very recent block Wiedemann approach of [KV01] to the structured determinant evaluation (where we incorporate the Chinese remainder algorithm, and randomized Newton's interpolation to speed up the computation in the case where the determinant nearly vanishes, that is exactly in the case we need, although our progress should also have general appeal and independent importance) and acceleration of the resultant computation by exploiting quasi-Toeplitz matrix structure, rational formulae for the toric resultant (in order to bound the output size), and evaluation/interpolation techniques [Too63] (cf. also [Ber02]).

1.2 The known and new results

Previous work on integer determinant evaluation includes [ABM99, EGV00, Kal02, KV01, Pan02a]; other works such as [BEPP99, PY01] focus on sign determination but do not improve upon the current record complexities. The best complexities for a general scalar matrix are due to [EGV00, KV01]; cf. also theorem 2.1 below. Let $O^*(f)$ indicate that the factors polylogarithmic in f are omitted. Then the known randomized bit cost bounds are in $O^*(m^{3+1/5}L)$ [KV01] and $O^*(m^2\gamma^{2/3}L)$, where m denotes the dimension of integer matrix M, L its maximum 2-norm of column (infinity) norm (our choice), and γ the number of arithmetic operations for multiplying M by a vector of scalars. We bound the bit complexity of computing $(\det M) \bmod s$, for s being the product of random primes, by $O^*(m\gamma q)$, where $q = \log s$, and by $O^*(m^{5/3}L^{2/3}\gamma^{2/3}q^{1/3})$. The recent works [Kal02] (which yields a bound of the order of m^3q) and [Sto02] do not apply efficiently to the quasi-Toeplitz structure and thus are not sufficient to support our improvements. For $s > 2|\det M|$ we can immediately recover $\det M$ from its value modulo s, so our bit cost bounds turn into the new bit cost record output sensitive bounds for computing $\det M$. We also derive the bit cost bounds in terms of L, m, s only (without γ); this improves the record bounds of [Kal02] by the factor of $(q/L)^{3/10}$, where $q := \lg(|\det M| + 2)$ denotes the bit size of the computed integer determinant. We extend all our results and new record bit cost estimates to uni- and multivariate matrices.

Another recent result [Sto02] considers univariate matrices and proposes a Las Vegas algorithm for their determinant computation with arithmetic complexity in $O^*(\mu d_1)$, where $O(\mu)$ bounds the arithmetic complexity of matrix multiplication and d_1 bounds the degree of the entries. Then again our estimates using structure are superior for resultant matrices.

Regarding resultant operations, we reduce bit complexity by almost an order of magnitude in terms of the resultant matrix dimension, which is the largest quantity involved in the bounds. Roughly speaking, the bit complexity of existing approaches is proportional to m^3. These results, unbeaten for years, remained a clear challenge to the algorithm designers. In the present paper we finally make a decisive step forward.

For evaluating a scalar resultant, the existing approaches exploit matrix structure [CKL89, EP02] to arrive at bit complexities in $O^*(m^3n^2DL)$, where n is the number of eliminated variables and D denotes the total degree of the resultant R thought of as a polynomial in the input coefficients. We derive the bound $O^*(m^2nDL)$, where L is the bit length of these coefficients when specialized. Alternatively, a Monte Carlo bound depending on the bit size q of the resultant's value is $O^*(m^2nq)$. Analogous improvements are obtained in the harder and highly important case where the input degenerates and a perturbation is applied. This improves upon available approaches. We mention only one of those, because it uses a different and more direct method: The problem is reduced to computing a characteristic polynomial with Monte Carlo bit complexity proportional to $m^{3+1/5}$, up to polylog factors for a general $m \times m$ matrix [Pan02a]. Even an improvement to m^3 would not beat our estimate in the quasi-Toeplitz case.

For expanding a univariate resultant $R(x)$, the algorithm of [Man93] is based on Kronecker canonical forms. It yields a bit cost bound in $O((m + \deg(\det M(x)))^3)$, under the condition that M is nonsingular as a matrix polynomial in x. Otherwise, the algorithm has complexity proportional to m^4. The fastest general algorithms have bit complexity in $O^*(m^3 n \deg(\det M(x)))$, typically relying on interpolation and matrix structure [CKL89, EP02]. Our algorithms support a bit cost bound, for arbitrary inputs, in $O^*(m^2 nDVL)$ where V is the actual degree of the univariate resultant. They also dramatically improve the known algorithms in the case of perturbed resultants. Similar improvements are obtained for expanding the n-variate u-resultant $R(u)$. This task has bit complexity proportional to $m^2 h$, where h denotes the number of nonzero terms in $R(u)$. Previous methods have complexity cubic in m [CKL89, EP02, Man93].

These results on resultant evaluation and expansion, coupled with the quotient formula of the (toric) resultant, immediately yield an improvement on the bit complexity of numerically approximating all isolated roots of a well-constrained algebraic system to a prescribed precision. Existing methods based on resultant formulations have *arithmetic* complexity cubic in m [CKL89, EP02, Mou98]. We employ the primitive-element method leading to an implicit representation of the output and, by combining all our techniques, we obtain *bit complexity* bounds which are roughly cubic in m.

1.3 Paper organization

This paper is organized as follows. The rest of this section overviews previous work and states our main results; previous work is also covered in the sequel, in particular in section 2 and the Appendix. Sections 2 and 3 focus on computing the determinants of integer and polynomial matrices and improve the known complexity bounds for structured matrices and in the cases of the computation modulo a product of random primes and output-sensitive algorithms. Section 4 formalizes the notion of resultant matrices, and section 5 accelerates their evaluation. Section 6 improves upon the existing algorithms for expanding resultants, and section 7 does the same for polynomial system solving.

2 Multivariate determinants preliminaries

This section introduces notation and details the main algorithms for determinant evaluation on which our improvements shall be based. Our second base for improvement, concerning rational formulae for the resultant, shall be introduced later.

The algorithms for determinant evaluation use *block Wiedemann's algorithm with structured preconditioning* [KP91, KS91, KV01, Wie86]. Our complexity bounds rely on fast univariate polynomial multiplication and/or on the Fast Fourier Transform (FFT), which is truly advantageous only for large inputs; otherwise, the classical or the Karatsuba divide-and-conquer algorithm may be preferable. On the other hand, none of the stated bounds relies on fast matrix multiplication, i.e., methods with complexity smaller than cubic in the dimension, which may be hard to exploit in practice. Still, both the existing and our methods can be coupled with fast matrix multiplication in order to reduce the asymptotic bounds.

Let us fix notation for the entire paper. We assume integer matrices or matrix polynomials. We also assume euclidean norms $\|\sum_i c_i x^i\|_2 = \|(c_i)_i\|_2 = (\sum_i \|c_i\|)^{1/2}$ for polynomials and vectors, and the consistent 2-norm $\|(M_{i,j})_{i,j}\|_2 = \max_{v:\|v\|=1}\{\|Mv\|_2\}$ for matrices, so

$$\|M\|^i \geq \|M^i\|. \tag{1}$$

Here $\|t\| = |t|$ for a scalar t. Alternatively, we may use the maximum or infinity norm for polynomials and vectors with $\|\sum_i c_i x^i\|_\infty = \|(c_i)_i\|_\infty = \max_i\{|c_i|\}$ and the consistent column norm for matrices $\|M\|_\infty = \max_{v:\|v\|=1}\{\|M_{i,j}v\|_\infty\} = \max_i \sum_j |M_{i,j}|$.

Let $\gamma(M)$ or γ denote the number of arithmetic operations required for multiplying a scalar vector by the matrix M. Unless otherwise stated, it suffices that γ bounds the complexity of vector multiplication on one side of the matrix. For a matrix polynomial $M = M(x)$, x denoting a variable or a set of variables, let $\gamma(M)$ denote the maximum $\gamma(M(\alpha))$ over the set of all values α of the variable(s) x. For an $m \times m$ matrix M, $m \leq \gamma \leq 2m^2 - m$ and $\gamma = o(m^2)$ for sparse and/or structured matrices.

We next present the known randomized algorithms and bit complexity bounds.

Theorem 2.1 [Pan02a, thm.5.1], cf. [KV01]. *Let M be an $m \times m$ matrix whose entries are polynomials in k variables with respective degrees less than $d_1, \ldots, d_k$, $k \geq 0$, and with integer coefficients. Let $L := \log \|M\|, \Delta :=$*

$d_1 \cdots d_k$, $g := 2/(k^2+4k+5)$. *Then* $\det M$ *can be computed by a Las Vegas algorithm with bit complexity bounded by*

$$O^*(m^{k+2+(k+3)g}\Delta L),$$

and

$$O^*(m^{k+2}\gamma^{2/(k+3)}\Delta L).$$

In particular, for scalar determinants we have the bounds $O^*(m^{16/5}L)$ *and* $O^*(m^2\gamma^{2/3}L)$, *whereas for univariate determinants these bounds become* $O^*(m^{19/5}d_1L)$ *and* $O^*(m^3\gamma^{1/2}d_1L)$.

Let us supply some details for deriving the above cost bounds, which are instrumental in extending them below. Theorem 2.1 is supported by the algorithm in [Pan02a] which extends the one in [KV01, sect.3], outlined in our Appendix.

The algorithm computes $\det M$ modulo sufficiently many distinct random primes $s_1, \ldots, s_u$ and then recovers $\det M$ by applying the *Chinese remaindering algorithm* (abbreviated as CRA); cf. [Zip93, vzGG99]. The random primes have relatively small length of the order of $\log(mL)$. The complexity of generating the primes is not studied explicitly, as is customary in such studies, because it is practically at acceptable level [vzGG99]. Under the requirement that $\log s > 1 + Lm > 1 + \log|\det M|$ for $s = s_1 \cdots s_u$, the algorithm either fails with a small probability or ensures computing the correct output at the claimed bit cost, dominated at the evaluation stage.

The bit cost of performing the entire algorithm is dominated by the bounds $O^*(m^3r^{k+1}\Delta L)$, $O^*((m^3/r)(m/t)^{k+1}\Delta L)$, and

$$O^*(m^{k+2}t^2\Delta L), \tag{2}$$

achieved at stages 2.2, 2.3, and 3, respectively. We are based on (1) and on the simple observation that $\Delta(M^h) \le \Delta(M)h^k$, for any positive $h \in \mathbb{Z}$.

Now we consider the three bounds above for the parameters r and t of our choice, which have to satisfy $1 \le r \le m/t$, $m \ge t \ge 1$ since they define baby steps / giant steps and blocking in Wiedemann's algorithm, respectively. The overall cost bound $O^*(m^{k+2+(k+3)g}\Delta L)$ is obtained when the three summands are made asymptotically equal. The first two summands are equal for $r := (m/t)^{(k+1)/(k+2)}$, and the last two for $t := m^{(k+3)/(k^2+4k+5)}$. These specifications satisfy the requirements on r and t.

The bounds in terms of γ are obtained by trivializing the baby steps / giant steps from stage 2, i.e. writing $r = 1$ trivializes stage 2. Then the bit cost at stage 2.3 is bounded by

$$O^*\left(\gamma m(\frac{m}{t})^{k+1}\Delta L\right). \tag{3}$$

Stage 3 is as before. So the overall complexity is bounded by the sum of (2) and (3) Then for $t := \gamma^{1/(k+3)}$, we obtain the overall bound $O^*(m^{k+2}\gamma^{2/(k+3)}\Delta L)$.

3 Improved determinant computation

The estimates in the following subsections are in γ and thus decrease for structured matrices. Only the last subsection deviates from this rule and presents output-sensitive bounds for unstructured matrices. We are going to extend theorem 2.1 and for completeness partly repeat its derivation, which formally the reader is not required to read.

3.1 Computation modulo a product of random primes for structured matrices

Our first contribution (in theorem 3.2) is the computation modulo a product of random primes which exploits matrix structure. Let M be an $m \times m$ matrix, with $\|M(x)\| = 2^L$, whose entries are polynomials in $x_1, \ldots, x_k$ with respective degrees less than $d_1, \ldots, d_k$, $k \ge 0$, and with integer coefficients. Let $\Delta := d_1 \cdots d_k$ and set $\Delta := 1$ for $k = 0$.

Lemma 3.1 *For a multivariate matrix* $M(x)$ *as above, we have*

$$\|\det M(x)\|_l \le \|M\|_2^m \le 2^{Lm}$$

for $l = 2$ *and* $l = \infty$.

Proof. We use the Kronecker substitution, namely $x_1 \to y, x_2 \to y^{d_1+1}, x_3 \to y^{d_1 d_2+1}, \ldots$. So we have $\det M(x) = \sum_{i=0}^{N-1} c_i y^i$ where y is a single variable and $N = 1 + d_1 + d_1 d_2 + \cdots + d_1 \cdots d_k$. Let ω be a primitive N-th root of unity and let

$$\Omega = (1/\sqrt{N})(\omega^{ij})_{i,j=0}^{N-1},\ \Omega^* = (1/\sqrt{N})(\omega^{-ij})_{i,j=0}^{N-1}$$

be the scaled unitary matrices of the forward and inverse Fourier transforms, where $\|\Omega^*\|_l/\sqrt{N} \le \|\Omega\|_2 = \|\Omega^*\|_2 = 1$. Then $(\det M(\omega^i))_{i=0}^{N-1} = \sqrt{N}\Omega(c_j)_{j=0}^{N-1}$, $(c_j)_{j=0}^{N-1} = (1/\sqrt{N})\Omega^*(\det M(\omega^i))_{i=0}^{N-1}$,

$$\|(c_j)_{j=0}^{N-1}\|_l = (1/\sqrt{N})\ \|\Omega^*\|_l \|\ (\det M(\omega^i))_{i=0}^{N-1}\|_l.$$

We have $\|\Omega^*\|_2 = 1$, $\|(\det M(\omega^i))_{i=0}^{N-1}\|_2 \le \max_i |\det M(\omega^i)|\sqrt{N}$, so

$$\|\det M(x)\|_\infty \le \|\det M(x)\|_2 = \|(c_j)_{j=0}^{N-1}\|_2 \le \max_i |\det M(\omega^i)| \le \|M\|_2^m = 2^{Lm}.$$

□

We shall operate modulo $s = s_1 \cdots s_u$, i.e. s is the product of u random primes sampled in $[a, am)$ with $\log(am) = c\log(mL)$, for a fixed $c > 0$; for a sufficiently large parameter u we have $q = \log s \le cu\log(mL)$. Some bad choices of a random prime may cause failure of the main algorithm, but this is unlikely for a sufficiently large constant c [KV01]. Remark that c can be fixed below 5, cf. [KV01]. We may increase the range if we need more primes. We assume $\log(am) = c\log(mL) \le q \le cLm\gamma/2$, which holds if $2u \le Lm\gamma/\log(mL)$, and under this condition we may still choose $s = 2(mL)^{cLm\gamma/(2\log(mL))} > 2\|\det M\|$, for $c\gamma > 2$, due to the lemma.

The next theorem extends the second bound of theorem 2.1 (in terms of γ) to computation of $(\det M) \bmod s$ where $\log(am) \le q = \log s \le cLm\gamma/2$.

Theorem 3.2 *Under the above assumptions,* $(\det M) \bmod s$ *can be computed by a Las Vegas algorithm with bit complexity in*

$$O^*\left(m^{k+1}\gamma^{\frac{2}{k+2}}\Delta q\right)$$

and in

$$O^*\left(m^{k+1}\left((Lm\gamma)^2 q^{k+1}\right)^{\frac{1}{k+3}}\Delta\right),$$

where $q = \log s$.

The first bound is smaller iff $q \le 2Lm/\gamma^{1/(k+2)}$. The cost bounds do not cover the generation of random primes. As we mentioned, this is a customary assumption as long as the cost is practically at acceptable levels [vzGG99]. This is the case here. By lemma 3.1, $q = mL$ bounds $\log\|\det M\|$. On the other hand, for q of the order of Lm, the second bound above is exactly the second bound in theorem 2.1 (see corollary 3.4).

Proof and Algorithm. The theorem is supported essentially by [KV01, sec.3] adjusted to computing modulo s; we use the extension of this algorithm to matrix polynomials [Pan02a]. As mentioned, according to [KV01], the choice of primes in the above range is unlikely to cause failure of the algorithm. The overall bit complexity is specified in [Pan02a] to be in

$$O^*\left(\gamma m(\frac{m}{t})^k \min\{q, \frac{Lm}{t}\}\Delta + t^2 m^{k+1} q\Delta\right), \tag{4}$$

for $t \in \mathbb{Z}$ chosen in $[1, m]$. The first summand corresponds to stage 2.3, where the baby steps / giant steps have been trivialized in order to be able to bound the complexity in terms of γ, and Lm/t is replaced by $\min\{q, Lm/t\}$. The second summand expresses the complexity of stage 3 where we replace mL by q. Other stages are dominated.

To support these bounds, we need to operate modulo s in $O^*(q)$ bit operations per arithmetic operation. This is achieved by Schönhage-Strassen's algorithm or by CRA. Practically the latter is preferred because of the considerable overhead of the former. Theoretically one may combine the two approaches to compute $\det M$ modulo several larger coprimes $s_1, \ldots, s_v$, each a product of several random primes in $[a, am)$, and then recover $\det M \bmod s$, $s = s_1 \cdots s_v$ via the CRA. The recovery's cost is dominated.

Now distinguish between the two cases above depending on whether $\min\{q, Lm/t\}$ is q or Lm/t. Pick t to make the two summands in the bound equal up to a constant factor, such that $t^{k+2} := \gamma/2$ or $t^{k+3} := cLm\gamma/(2q)$, respectively. So the claimed bit cost bounds hold. Furthermore, in the first case, the hypothesis that $q \le Lm/t$ implies that $q \le 2Lm/\gamma^{1/(k+2)}$. In the second case, the hypothesis that $q > Lm/t$ implies $q > 2Lm/\gamma^{1/(k+2)}$, for the chosen value of t.

In both cases, the hypothesis $t \le m$ is satisfied for $\gamma \le 2m^{k+2}$, where the latter bound holds true even for $k = 0$. Similarly, $1 \le t$ holds because $\gamma \ge 2$ and $2q \le cLm\gamma$ by assumption.

In extending the bounds from the scalar to the multivariate case, it is important to bound the degree in the k variables of the product of M with a vector, repeated m/t times. Initially, the vector has only scalar entries; at the end its degree in the i-th variable is at most $(m/t)d_i$. Hence the product of degrees in the final vector product is $(m/t)^k\Delta$. □

Corollary 3.3 *The above bounds specialize to* $O^*(m\gamma q)$ *and* $O^*(m^{5/3}L^{2/3}\gamma^{2/3}q^{1/3})$ *for scalar determinants, and to* $O^*(m^2\gamma^{2/3}d_1q)$ *and* $O^*(m^{5/2}L^{1/2}\gamma^{1/2}q^{1/2}d_1)$ *for univariate determinants. The former bounds are superior iff* $q \le 2Lm/\gamma^{1/(k+2)}$.

Corollary 3.4 *If* $s > 2|\det M|$, *theorem 3.2 can be applied to the evaluation of* $\det M$ *yielding the bit cost bound* $O^*(m^{k+2}L\gamma^{2/(k+3)}\Delta)$, *which is the second bound of theorem 3.2 for* $q = cLm$ *and for a constant* c *and thus is exactly the second bound of theorem 2.1. In particular, for scalar determinants, if* $s > 2(\sqrt{m}2^L)^m$, $q > 1+m(L+\log(\sqrt{m})) > 2Lm/\gamma^{1/(k+2)}$, *then* $s > 2|\det M|$ *and the bound becomes* $O^*(m^2L\gamma^{2/3})$.

3.2 Output sensitive bit complexity for structured matrices

This section employs the previous approach, based on theorem 2.1 and 3.2 and exploiting matrix structure, with the additional goal of achieving output-sensitive improvements of the known algorithms. In our iterative algorithms for polynomial systems, we may compute $\det M$ where it nearly vanishes, and then the output sensitive approach enables a substantial speedup, because the bit cost of determinant evaluation dominates the CRA cost. This leads to an improvement upon (output insensitive) algorithms whose complexity depends on static *a priori* bounds on $|\det M|$, e.g. [ABM99, KV01, Pan02a]. Static bounds on $|\det M|$ usually rely on Hadamard's inequality, which is notorious for being excessively high in general [ABM99], and even more so when $\det M$ may nearly vanish.

Practically, as we noted, $s_1, \dots, s_u$ should be random primes sampled from a fixed range, then the number of residues required in the CRA is roughly proportional to a bound on the bit size of the output value v. Reconstructing v by Lagrange's deterministic scheme has bit complexity quasi-linear in this bound, versus $O(q^2)$ by Newton's incremental method, with q expressing the *actual* bit size of v, namely $q = \log_2(|v| + 2)$. The latter method was applied in determinantal computations in [BEPP99]. Hence Newton's CRA in [BEPP99] is an output-sensitive algorithm (cf. [MC93]). It is a Monte Carlo algorithm because it stops when the computed value may still be different from the desired correct output but only with a low probability. More precisely, for primes s_i, write $v_k = \sum_{i=0}^{k} b_i s_1 \cdots s_i, b_i = (v - v_i)(s_1 \cdots s_i)^{-1} \bmod s_{i+1}$. By sampling s_i uniformly from a set S, we obtain $\text{Prob}\{v_k = v \text{ if } v_k = v_{k-1}\} \ge 1 - 1/|S|$, whereas the bound $s_1 \cdots s_k > 2|v|$ implies deterministically that $v_k = v$.

Let us use an *a posteriori* Monte Carlo bound in an output-sensitive way. Recall that the s_i are random primes from $[a, am)$. Let $q_j := \log s_j$ and $q^{(j)} := \sum_{i=1}^{j} q_i$. We compute $\delta_j := (\det M) \bmod s^{(j)}$, $s^{(j)} := s_1 \cdots s_j$, $j = 1, 2, \dots, h$ for h such that $\delta_{h-1} = \delta_h$.

Corollary 3.5 *With high probability* $\delta_{h-1} = \det M$, *using the above notation. To increase the probability of success, we may require that* $\delta_{h-b} = \delta_h$ *for some fixed* $b > 1$. *Then, with high probability, the computation of* $s^{(j)}, j = 1, \dots, h-1$, *and hence of* $\det M$, *has bit complexity bounded by the expressions in theorem 3.2 for* $q = q^{(h)}$, *where the first bound is superior iff* $q^{(h)} \le 2Lm/\gamma^{1/(k+2)}$.

Proof and Algorithm. Combine the algorithm of theorem 3.2 for $q = q^{(h)}$ with Newton's CRA to compute the δ_j. Concerning the probabilities see [BEPP99, Emi98, Kal02]. A delicate point in our present proof is that $q^{(h)}$ is not known until we observe that $\delta_{h-b} = \delta_h$ for $b \ge 1$, and the value $q^{(h)}$ affects the choice of the parameter t. If we blindly extend the algorithm supporting theorem 3.2, we may have to recompute $(\det M) \bmod s_i$ for the current primes s_i as soon as we change t. This could increase the bit cost by the factor of the order of u (the number of primes s_i), which can be of the order of Lm. A simple way out, however, is not to change t until the new value of q is doubled versus the last time it defined t. Then the overall number of distinct t is $O(\log(Lm))$ and the increase of the bit cost by the factor of $\log(Lm)$ is immaterial under the O^* notation. □

3.3 General output-sensitive bounds

Let us now apply the algorithm in [KV01] in the multivariate case but with baby steps / giant steps. We do not express complexity in terms of γ here, hence the usefulness of the next theorem for structured resultant matrices is

limited. Nonetheless, the result is presented for completeness, since it is of independent interest and achieves record output-sensitive complexity estimates.

Theorem 3.6 *Suppose $q \geq L$ and $q \geq \log(c\log(mL))$ with the above notation (cf. theorem 2.1). Then* $\det M$ *can be computed by a Monte Carlo algorithm with the bit complexity bounded by:*
(a) $O^*(m^{k+1+(2k+5)g}L^{(k+2)g}q^{1-(k+2)g}\Delta)$,
where $g := 2/(k^2+4k+5)$;
(b) $O^*(m^{k+1+(k+4)h}L^h q^{1-h}\Delta)$,
where $h := 2/(k^2+3k+4)$;
(c) $O^*(m^{k+1+(k+2)f}q\Delta)$,
where $f := 2/(k^2+2k+2)$. *These bounds improve theorem 2.1 for* $q \leq Lm$, $q \leq Lm^{k(k+2)/(k^2+3k+3)}$, $q \leq Lm^{k^2f/2}$, *respectively.*

Proof. The estimates in section 2, namely (2) and the ensuing discussion, rely on using the bit length of the orders of $Lr, Lm/t$, and Lm at stages 2.2, 2.3, and 3 of the algorithm, respectively. Using the output sensitive CRA (with dynamic choice of primes separately for each of stages 2.2, 2.3, and 3), we may replace the bit length bounds by q wherever $q < Lr, q < Lm/t$, and $q < Lm$ and then we may still satisfy the constraints $1 \leq r \leq m/t$, $1 \leq t \leq m$. The bit complexity of stages 2.2, 2.3, and 3 becomes $O^*(m^3r^k \min\{q, rL\}\Delta)$, $O^*((m^3/r)(m/t)^k \min\{q, Lm/t\}\Delta)$, and $O^*(m^{k+1}t^2q\Delta)$, respectively. As in the proof of corollary 3.5 we change the bound on the bit length of q only when this at least doubles it, and thus we limit the number of changes to $O(\log(Lm))$.

To prove (a), we equalize the three stages' bounds by choosing

$$r := (m/t)^{(k+1)/(k+2)} \leq m/t,\ t := (m^{(2k+5)/(k+2)}L/q)^{(k+2)g/2}.$$

Then $t \leq m \Leftrightarrow L/q \leq m^k$, which holds for $k \geq 0, L \leq q$. With no extra hypotheses, we may simply replace $\min\{q, rL\}$ by rL and $\min\{q, Lm/t\}$ by Lm/t, even if $q < rL, q < Lm/t$, thus obtaining the bounds encountered in the context of theorem 2.1 for stages 2.2 and 2.3. The first two summands are equal due to our choice of r, and the last two summands become equal for our choice of t.

For (b), if we write $r := m^2/t^{k+2}, t := m^{(k+4)h/2}(L/q)^{h/2}$, then the hypothesis $q \leq Lm^{k(k+2)/(k^2+3k+3)}$ is equivalent to $q \leq Lm/t$. Therefore, we replace $\min\{q, Lm/t\}$ by q. We also replace $\min\{q, rL\}$ by rL, which is always possible. For the chosen values of r and t, we have $r \leq m/t$ because $r/(m/t) = m/t^{k+1}$, which equals

$$m^{1-(k+1)(k+4)/(k^2+3k+4)}\ (q/L)^{(h/2)(k+1)}.$$

This is bounded by $m^{-kh}(q/L)^{(k+1)h/2}$. For $k = 0$, we have $q = L$, so $r/(m/t) \leq 1$. Otherwise, $k \geq 1$. Then substitute $q/L \leq m/t$ and obtain that

$$\frac{r}{m/t} \leq m^{(1-k)h/2} / t^{(k+1)h/2} < 1.$$

Now observe that the choice of r makes the last two summands equal. Similarly, the choice of t makes the first and third summands equal. For this specialization, $t \leq m$ can be established by using $tq \leq Lm$ and $L \leq q$ and $t \geq 1$ by using $q \leq mL$.

For (c), let us write $r := (m/t)^{k/(k+1)} \leq m/t$, $t := m^{(k+2)f/2} \leq m$; these choices satisfy $t, r \geq 1$. Then the hypothesis $q \leq Lm^{k^2f/2}$ is equivalent to $q \leq rL$. Since $r \leq m/t, q \leq rL$ we have $q \leq mL/t$, so we can replace $\min\{q, rL\}$ and $\min\{q, Lm/t\}$ by q. The chosen value of r makes the first two summands in the overall bound equal. The choice of t makes the last two summands equal. □

Corollary 3.7 *The theorem's bounds specialize as follows. For $k = 0$: (a) $O^*(m^3L^{4/5}q^{1/5})$, (b) $O^*(m^3L^{1/2}q^{1/2})$ [Kal02], and (c) $O^*(m^3q)$. For $k = 1$: (a) $O^*(m^{3+2/5}L^{3/5}q^{2/5}d_1)$ (b) $O^*(m^{3+1/4}L^{1/4}q^{3/4}d_1)$, (c) $O^*(m^{3+1/5}qd_1)$.*

Proof. $k = 0 \Rightarrow g = 2/5, h = 1/2, f = 1$. $k = 1 \Rightarrow g = 1/5, h = 1/4, f = 2/5$. Substitute these values in the bounds of theorem 3.6. □

4 Resultant matrices

This section reviews matrix formulae for the resultant. For further information see [CKL89, CLO98].

Consider a system of polynomials $f_0, \ldots, f_n \in K[x_1, \ldots, x_n]$, i.e., in n affine variables, with indeterminate coefficients; typically $K = \mathbb{Z}$ (or $K = \mathbb{Q}$). Then their *resultant* R is an irreducible polynomial in these indeterminates, whose vanishing provides a necessary and sufficient condition for the existence of common roots of the system in a specified variety. For the classical resultant of homogeneous polynomials in $n+1$ variables, this variety is the projective space $\mathbb{P}^n_K$. For the *toric resultant*, it is the toric variety obtained as the closure of the torus $(\overline{K} - \{0\})^n$ under the Veronese maps of certain monomials in a projective space of dimension usually larger than n, where $\overline{K}$ stands for the algebraic closure of K. In toric elimination theory, the polynomials can have integer exponents and each polynomial is characterized by its support (nonzero monomials) rather than its total degree.

Let us think of the f_i having *symbolic* coefficients. The resultant R is a homogeneous polynomial in the (symbolic) coefficients of each f_i, with integer coefficients. Its degree in the (symbolic) coefficients of f_i equals the generic number of common roots of the other n polynomials in the corresponding variety. This is given by Bézout's number or the mixed volume $\mathrm{MV}_j(\cdot)$, where we consider mixed volumes of sets of j polynomials in j affine variables. Mixed volume captures the sparsity of the equations, since it depends solely on their nonzero terms. It is more general than Bézout's number in that it reduces to the latter for n dense homogeneous polynomials in $n+1$ variables [CLO98]. Hence the total degree of R in the input coefficients is

$$D := \sum_{i=0}^{n} \mathrm{MV}_n(f_0, \ldots, f_{i-1}, f_{i+1}, \ldots, f_n). \tag{5}$$

There are two types of *resultant matrices* used to compute the resultant itself. Those of Bézout-type are discussed at the end of section 6 briefly. Most of this paper focuses on *Sylvester-type* matrices, specified by means of generic polynomials $g_0, \ldots, g_n$ such that the map

$$[g_0, \ldots, g_n] \mapsto [g_0, \ldots, g_n]\ M = \left[\sum_{i=0}^{n} g_i f_i\right]$$

has two properties: first, it is surjective for generic f_i and, second, the dimensions of the domain and the range are equal. Generic polynomials can be thought of as having indeterminate coefficients. Then M is an $m \times m$ Sylvester-type resultant matrix such that $R \mid \det M$ and $\det M \not\equiv 0$. In the case of dense homogeneous polynomials, this is the classical Macaulay matrix. Matrix construction is independent of the coefficient values and can thus be conducted off-line; its complexity can be asymptotically smaller than that of manipulating the matrix for system solving.

M is *quasi-Toeplitz*, i.e., its entries depend only on $\alpha - \beta$, where α, β belong to two subsets of $\mathbb{Z}^n$ which index, respectively, the rows and columns of M. The most relevant property of such matrices is that, by [EP02, thm.5.6] (cf. [CKL89]), for a vector v and M both filled with scalars, computing $v^T M$ and Mv takes $O^*(mn + n\sqrt{d})$ arithmetic operations, where d is the maximum degree of f_i in any variable. Assuming $m \geq \sqrt{d}$, the time complexity becomes $O^*(mn)$; in practice, we usually have $m \gg d$. Hence, by Wiedemann's algorithm with structured preconditioning and [EP02], the arithmetic complexity of computing $\det M$ is $O^*(m^2 n)$.

We may assume, $m > n$, otherwise the polynomial system degenerates and its resultant can be defined by fewer than $n+1$ polynomials. In the worst case, m grows exponentially with respect to n; typically, $m \gg D$. Our algorithms heavily rely on the recent *quotient formula* of [D'A02]:

$$R = (\det M)/(\det S),$$

which extends Macaulay's classical result to the toric case, with S being a submatrix of the Sylvester-type resultant matrix M. This formula gives a general means for computing R exactly, and leads to output-sensitive bounds. When we specialize the input coefficients in K, with $K = \mathbb{Z}$, the resultant is in $\mathbb{Z}$ even though it is a ratio of two determinants. Furthermore, $s = \dim S < m = \dim M$ and $m - s = D$; $m \gg D \Rightarrow s \ll m$. The entries of M are integers of length ℓ.

To use resultants for finding all isolated roots of a well-constrained polynomial system with specialized coefficients, there are two approaches. The first considers $n+1$ input polynomials in $x_1, \ldots, x_{n+1}$, and regards this system as overconstrained over $K = \mathbb{Z}[x_{n+1}]$, where x_{n+1} is known as the *hidden variable*. No information is available *a priori* on the distribution of x_{n+1} in M [Man93]. Since the input coefficients are specialized, $R(x_{n+1})$ is a univariate polynomial with degree

$$V := \mathrm{MV}_{n+1}(f_0, \ldots, f_n)$$

in x_{n+1}. Its coefficients have bit length $D\ell$, due to the quotient formula in [D'A02] and definition (5). There is no a priori relation between V and D but it is safe to assume $\log V = O(D)$. It is always the case that $V < m$. Some examples, for the cyclic-N family, are the following, where $N = n+1$ in our notation: for $N = 4, 5, 6, 7$, we have $V = 16, 70, 156, 924$ and, respectively, $m = 25, 147, 851$ and more than 3000. See [EC95] for the definition of the cyclic-N systems and more examples on the relation of m and V.

The second way to arrive at an overconstrained system, when one is given n equations, is by adding a linear polynomial f_0 with indeterminate coefficients $u_1, \ldots, u_n$; they play the role of the hidden variables. Then R is a polynomial in these variables, where all coefficients are specialized, and is known as the *u-resultant*. In certain cases, we shall consider the homogenized u-resultant, with homogenizing variable u_0. Hence the hidden variables are $u = (u_1, \ldots, u_n)$ or $u = (u_0, \ldots, u_n)$. The u-resultant factorizes as $R(u) = C \prod_i L_i(u)^{e_i}$, where the L_i are linear, $\sum_i e_i = V_0 := \mathrm{MV}_n(f_1, \ldots, f_n)$, and C is independent of u; notice that $V_0 < D$. The polynomial $R(u)/C$ is also known as the Chow form of $f_1, \ldots, f_n$ [CLO98]. In $R(u)$, the coefficients' bit length is bounded by $D\ell$ as well as V_0 times the bit length of the roots (for the output-sensitive bound).

Degeneracies constitute the Achilles' heel of Sylvester-type matrices, including Macaulay's matrix, because these matrices are constructed for indeterminate coefficients. They must eventually be specialized to their input values, and this may make the determinant vanish identically. In addition, it is possible that the resultant polynomial vanishes, due to the existence of an infinite number of common roots, thus giving no information on the isolated roots. This is a degenerate case, since most applications still require determining all isolated solutions. The proposed linear perturbations [Can90, DE01, MC92] yield a *projection operator* as the trailing coefficient of $\det M(\epsilon)/\det S(\epsilon)$, which vanishes at the isolated roots, thus allowing their computation.

5 Resultant evaluation

This section studies evaluation of the scalar resultant. Let L stand for the matrix entries' bit length in general. This slightly differs from $L = \log \|M\|$ as used before, but by at most the additive term $\log m$, so we slightly abuse the notation for simplicity. When M contains integers, then $L = \ell$, the input data length. When M is univariate with d_1 bounding the degree of each entry, then, after specializing the variable to an (integer) value of fixed bit size, $L = O(d_1 + \ell)$. In the notation of section 2, we consider only two cases: $k = 1, d_1 \geq 1$ and $k = n, d_1 = \cdots = d_n = 1$.

Problem 5.1 *Compute the value of the scalar resultant R. If the resultant is a polynomial in some hidden variables, then suppose these variables have been specialized. In either case, L stands for the matrix entries' bit length.*

Lemma 5.2 *If $\gamma(A)$ denotes the arithmetic cost of multiplication of a matrix A by a vector and A' is a submatrix of A, then $\gamma(A') \leq \gamma(A)$.*

Let $\mu_A(X)$ and $\mu_B(X)$ represent asymptotic arithmetic and bit complexities, ignoring polylog factors, for computing X. For instance, when $X = R, R(x_{n+1})$ or $R(u)$, the respective complexities denote those for computing the specialization of resultant R, or the uni- or multi-variate resultant polynomial $R(x_{n+1})$ or $R(u)$.

In particular, $\mu_A(R) = O^*(m^2 n)$, which follows simply since $\gamma(M) = O^*(mn)$ by [EP02]. Notice that, by using Tellegen's reversion circuit theorem ([BCS97, thm.13.20]), $\gamma(M)$ bounds the asymptotic cost of pre- and post-multiplication under our computational model. We justify the application of Tellegen's theorem in the appendix.

Theorem 5.3 *With the above notation, a randomized Las Vegas algorithm solves problem 5.1 with bit complexity $\mu_B(R) = O^*(m^2 nDL)$. A Monte Carlo algorithm solves the same problem with bit complexity $\mu_B(R) = O^*(m^2 nq)$, where $q := \lg(\|R\| + 2)$ expresses the actual bit size of the specialized resultant R.*

Proof and Algorithm. The algorithm is based on the CRA. It requires evaluation of $R \bmod s_i$, for $q+1$ primes s_i of logarithmic length. The discussion of the previous section implies $q = O(\log V_{(0)} + DL)$, where $V_{(0)}$ stands for V or V_0, depending on whether we deal with the case of a hidden variable x_{n+1} or the u-resultant. But $\log V = O(D)$ and $V_0 \leq D$, as explained in the previous section. Therefore, $q = O(DL)$.

Apply corollary 3.3 with primes s_i such that $\log s_i = O(\log(m \log \|M\|))$ for scalar matrices M and S. The bit operation cost of each evaluation of $\det M$ is $O^*(m\gamma \log(m \log \|M\|)) = O^*(m^2 n \log\log \|M\|)$, by using the first part of the corollary for $q = \log s_i$. S is a submatrix of M, thus, by lemma 5.2, its determinant is computed within the same complexity as M. So the bit cost of the evaluation phase is $O^*(m^2 nq)$.

Reconstructing the value R uses Lagrange's deterministic scheme with bit complexity quasi-linear in DL. This is dominated by the cost of evaluations since $DL = O^*(m^2 n)$. The latter equation follows from the fact that $D \ll m$, and we safely assume that L is not too large.

For the Monte Carlo version, $O(q)$ evaluations determine the overall complexity. The reconstruction phase has bit cost in $O(q^2)$ by Newton's interpolation with early termination [BEPP99]. □

The rest of this section covers linearly perturbed resultant matrices. The numerator is denoted by $M(\epsilon) = M_0 + \epsilon M_1$ where $\det M(\epsilon) = \epsilon^k D_k + \cdots + \epsilon^m D_m$. The bit length of the integer entries of M_1 is assumed the same as that of the input M_0, denoted L. The perturbation guarantees that $D_k \neq 0$ for some $m \geq k \geq 0$; typically, $m \gg k$. The perturbed denominator is $\det S(\epsilon) = \epsilon^s S_s + \cdots + \epsilon^t S_t$, with $S_t \neq 0$ such that $s \geq t \geq 0$. By the divisibility of the (perturbed) determinants, $t \leq k$. The degree and coefficient length of the projection operator are bounded by the respective quantities of the resultant.

It is possible that M, S contain indeterminate(s) x_{n+1} or $u_1, \ldots, u_n$. In this section, all indeterminates are specialized such that $D_k S_t \neq 0$.

Problem 5.4 *Assume the resultant evaluates to zero. Compute the value of the trailing coefficient of the perturbed resultant, namely D_k/S_t, when both numerator and denominator matrices are perturbed with respect to $\epsilon \to 0^+$. $D_k \neq 0$ and $S_t \neq 0$ are the trailing ϵ-coefficients in $\det M(\epsilon)$ and $\det S(\epsilon)$, respectively.*

It is known that k can be found by *binary search* in $[0, m]$ with bit complexity $m \log m \mu_A(\det M) = O^*(m\mu_A(\det M))$. [DE01, lem.5.1]. It is realistic to suppose $k = O(L\sqrt{m/n})$:

Lemma 5.5 *There is a Las Vegas randomized algorithm that determines k with bit complexity in $O^*(m^2nDLk)$, and t in $O^*(m^2nDLt)$. There is a Monte Carlo algorithm that determines k with bit complexity in $O^*(m^2nk)$, and t in $O^*(m^2nt)$.*

Proof and Algorithm. The algorithm computes $O(\log k)$ univariate polynomials $\det M(\epsilon) \bmod \epsilon^h$, for $h = 1, 2, 4, 8, \ldots$. Each such computation has bit complexity bounded by that of computing the scalar $\det M$ multiplied by $O^*(h)$. Then it suffices to apply theorem 5.3. The same algorithm works for computing t from matrix $S(\epsilon)$.

In the Monte Carlo approach, first randomly specialize $\epsilon \mapsto r_0$. Clearly, $h \leq k \Rightarrow (\det M(r_0)) \bmod r_0^h = 0$, whereas $h > k \Rightarrow (\det M(r_0)) \bmod r_0^h \neq 0$, with a high probability. Supposing r_0 is sufficiently small compared to $\det M(r_0)$, we deduce that $(\det M(r_0)) \bmod r_0^h$ is uniformly distributed in $[0, r_0^h)$ so that the probability of error is $1/r_0^h$. Therefore, $(\det M(r_0)) \bmod r_0^h \neq 0$ implies $h > k$, otherwise the algorithm decides that $h \leq k$; the test can be repeated with different random values of r_0 in order to decrease the error probability.

The algorithm performs a constant number of tests on whether $\det M(r_0) = 0 \bmod r_0^h$, for each h where $h = 1, 2, 4, 8, \ldots$ takes $O^*(\log k)$ values. Each test is performed modulo an integer s of bit size $h \leq k$. So we can apply the first part of corollary 3.3, due to the bound on k. Hence the overall cost is $O(m\gamma k \lg k)$, where $\log s = \Theta(k)$. Finally, use $\gamma = O^*(mn)$.

The algorithm supports an analogous complexity bound for computing t, by replacing k by t. □

The CRA-based algorithm in [DE01] can be adapted to the quotient formula to support bit complexity $O^*((k-t)\mu_A(\det M)mL) = O^*(km^3nL)$ for problem 5.4 provided k, t have been computed. Without knowing k, t, it is still possible to solve this problem by the approaches discussed in the next section, all of them with bit complexity proportional to m^3. We next show the solutions with a smaller bit cost and with no *a priori* knowledge of k, t.

Corollary 5.6 *There exists a Las Vegas algorithm for problem 5.4, with bit complexity $O^*(km^2nDL)$. There exists a Monte Carlo algorithm for the same problem, with bit complexity $O^*(m^2nq)$, where $q := \lg(|D_k/S_t| + 2)$.*

Proof and Algorithm. For the Las Vegas method, we perform the computation with symbolic ϵ, modulo ϵ^h; this adds the factor $O^*(h)$ to the complexity estimates for determinant evaluation, for each candidate $h = 1, 2, 4, 8, \ldots$, until h reaches or exceeds $k - t$, which is the degree corresponding to the trailing coefficient. There are about $\log(k-t)$ evaluations. Therefore the bit complexity equals that of evaluating R, multiplied by $O^*(k-t)$. The claim follows by dropping the dependence on t for simplicity.

The Monte Carlo approach first computes k, t by applying the algorithm of the previous lemma separately to the numerator and denominator. We then apply the CRA with $O(q)$ evaluations. Then corollary 3.3 gives an overall bound of $O^*(m^2nq)$ for the Monte Carlo algorithm. The latter cost dominates the complexity of computing k, t, because we may suppose $k = O^*(\log |D_k/S_t|)$. □

6 Resultant expansion

This section studies the problem of computing the resultant as a uni- or multi-variate polynomial, assuming it is a nonzero polynomial.

Problem 6.1 *Assume the entries of resultant matrix M lie in $\mathbb{Z}[x_{n+1}]$, with coefficients of size ℓ. Then $R(x_{n+1}) = \det M(x_{n+1})/\det S(x_{n+1})$. Compute the polynomial $R(x_{n+1})$, of degree V, in the monomial basis.*

Corollary 6.2 *Problem 6.1 can be solved by a Las Vegas algorithm with bit complexity $O^*(m^2nVD\ell)$ and by a Monte Carlo algorithm with bit complexity $O^*(m^2nVq)$, where q expresses the actual bit size of the specializations of $R(x_{n+1})$.*

Proof and Algorithm. We use Toom's standard evaluation-interpolation over $V+1$ integer values (see [Too63], [Ber02]). Each resultant value is computed with bit complexity $\mu_B(R)$, given by theorem 5.3, with $L = \ell$. The interpolation phase reduces to solving a transposed Vandermonde system, with arithmetic complexity $O^*(V)$ [Pan01], and its cost is dominated by the evaluation cost. If, instead, we apply the Monte Carlo version of theorem 5.3, we obtain the bound in terms of q. □

Next consider the u-resultant. It is possible to factor out of $\det M(u)$ the minor $\det Q$, where Q is the largest square submatrix of $M(u)$ filled with scalars. In general, Q contains S as a proper submatrix.

Problem 6.3 *Compute $R(u) = \det M(u)/\det Q$, of total degree V_0, in the monomial basis. Q is a submatrix of M containing scalars, such that $\dim Q = m - V_0$.*

It is possible to apply theorem 2.1 for $k = n+1$ variables yielding a bound proportional at least to m^{n+3}. This bound grows as much as exponentially in n^2 because m may be exponential in n. Our bounds reduce this exponent of m.

Corollary 6.4 *If h' is the actual support cardinality of $R(u)$ and q the maximum bit length of its specializations, then a Monte Carlo algorithm for problem 6.3 has bit complexity $O^*(h'nV_0\mu_B(R)) = O^*(h'm^2n^2V_0q)$.*

Proof and Algorithm. Zippel's sparse interpolation algorithm [CKL89, Zip93] leads to an output-sensitive Monte Carlo solution: its cost depends on the *actual* support cardinality, denoted by $h' \le h$. The bottleneck is the evaluation stage, which requires $O^*(h'nV_0)$ integer values, each computed with complexity $\mu_B(R)$ as in theorem 5.3 by a Monte Carlo algorithm. The interpolation cost is dominated by the evaluations. □

The algorithms discussed above can be readily extended to the case that the resultant polynomial is identically zero. Then, one wishes to compute the trailing coefficient of the perturbed resultant in one or $n+1$ hidden variable(s) when both numerator and denominator are perturbed by $\epsilon \to 0^+$. The sought polynomial, known as a projection operator, equals $D_k(x_{n+1})/S_t(x_{n+1})$ or $D_k(u)/S_t(u)$, respectively. The precise algorithms and their complexities are based on those supporting corollaries 5.6 and 6.4, but are omitted here for the sake of being concise.

In computing the Bézout-type resultant matrices, the bottleneck is expanding the determinant of an $(n+1)$-dimensional matrix, which contains the discrete differentials of the input polynomials f_i, $i = 0, \ldots, n$, in $2n$ variables [Mou98, Zip93]. The coefficients have bit length ℓ and, when all variables are specialized to scalars of fixed length, the bit size of the matrix entries is $L = O(dn + \ell)$, where d bounds the degree of f_i in each variable. The number of terms in the determinant is bounded by $n!d^n = O((nd/e)^n)$. Hence, a sparse interpolation method like those used above has bit complexity in $O^*((nd/e)^n(d+\ell))$ by theorem 2.1.

7 Algebraic system solving

This section first examines methods for numerical approximation of a specific coordinate of the zeros, e.g. the $(n+1)$-st coordinate, by solving the univariate resultant $R(x_{n+1})$. The second problem under examination is to compute the zeros of a zero-dimensional polynomial system, which is equivalent to factoring the u-resultant $R(u)$. These problems signal the involvement of numerical approximation algorithms. This is typical for multivariate polynomial root-finding: symbolic techniques are used at the first stage, which can be viewed as preconditioning, followed by numerical approximation techniques at the final stage.

Let us consider the case of a hidden variable x_{n+1}. If we have computed the coefficients of $R(x_{n+1})$ with respect to the monomial basis, we may solve the next problem by a variety of available numerical methods (e.g., [BP03, McN93, Pan97, Pan02b] and their references), but practically one may prefer to rely on computing polynomial values rather than the coefficients. Alternative exact-computation methods for root isolation are too expensive.

Problem 7.1 *When $R(x_{n+1}) = \det M(x_{n+1})/\det S(x_{n+1})$, approximate all (or some of) its zeros with output error tolerance $\tau = 2^{-b}$, without expanding it in the monomial basis. This yields one coordinate of all common roots of an algebraic system.*

Most popular polynomial root finders recursively update the current approximations to all (or one) root(s) of $R(x_{n+1})$ based on recursive evaluation of $R(x_{n+1})$ and possibly $R'(x_{n+1}) = dR/dx_{n+1}$ at these approximation points. Weierstrass' approach approximates all roots. It uses a multidimensional Newton iteration for solving Viete's system. Its variants include Durand-Kerner's, Aberth's, Farmer-Loizou's, Maehly's, and Werner's algorithms. They converge rapidly for any input polynomial as is testified by the decades of extensive practical computations. Hence we may practically assume that the number of iterations by these algorithms is expressed by $O(\log b), \tau = 2^{-b}$ denoting the root coordinates error tolerance, even though only proofs of the superlinear local convergence are available. At every iteration, the complexity is dominated by the cost of computing the values of R (in Durand-Kerner's) as well as R' (in the other algorithms) at $O(V)$ points. Initial approximations to the roots can be chosen heuristically; a customary choice consists of equally spaced points on a large circle centered at the origin.

Alternatively, our resultant evaluation techniques can be combined with Jenkins-Traub's, modified Laguerre's, or modified Newton's algorithms for computing a single root. None of these algorithms offers any guarantee of fast global convergence for high-degree polynomials, but their current versions converge quite well in practice. These algorithms can be recursively extended to the next roots via implicit deflation.

To apply the results of previous sections, we need to scale the variable when the iterative procedure is near the root in order to make the matrix entries integral. To approximate the root within $\tau = 2^{-b}$, we have to scale the matrix by τ. Then the bit precision factor $\log|\det M|$ in our cost bounds grows by an additive term of at most mb. The l factor grows by an additive term b which is absorbed in the $O^*()$ notation.

Theorem 7.2 *A Las Vegas iterative method, based on the Durand-Kerner algorithm, solves problem 7.1 for all roots by using $O^*(m^2nDV(\ell+mb))$ bit operations. A Monte Carlo algorithm for the same problem uses $O^*(m^2nV(q+mb))$ bit operations, where $q := \lg(\|R(x_{n+1})\| + 2)$. If only a subset of ρ roots must be output, then the Jenkins-Traub algorithm yields a Las Vegas and a Monte Carlo method with the respective bit costs above multiplied by ρ/V.*

Proof and Algorithm. The number of iterative steps of a rootfinder is $O(\log b)$. For evaluating R, we may use the Las Vegas algorithm supporting theorem 5.3 with complexity $\mu_B(R)$, with $L = \ell + mb$. Then the overall cost is $O^*(V\mu_B(R)\log b)$ bit operations. This yields a bound of $O^*(m^2nDV(\ell\log b + mb))$ bit operations. Since $\ell\log b + mb = O^*(\ell + mb)$, we arrive at the claim.

We may also apply the Monte Carlo algorithm of theorem 5.3. □

In order to use algorithms that require the derivative values, we may represent the Las Vegas algorithm of theorem 5.3 by a *straight-line program (SLP)*, i.e., without branching. SLPs define a polynomial by the black box for its (and its derivative) evaluation rather than its coefficients. Then, evaluating R' has the same cost (up to an extra factor of 4) as evaluating R [BCS97]. This is needed in all algorithms of the Weierstrass type except Durand-Kerner's, and those of Newton type, except Jenkins-Traub's. In fact, Laguerre's method requires second order derivatives, which multiplies the complexity by 16, but offers convergence of the 4th order.

Corollary 7.3 *All algorithms of the Weierstrass or Newton type mentioned above yield Las Vegas methods for solving the respective problems of the previous theorem. Their bit complexity in the SLP model is equal to that of the respective Las Vegas algorithm of the previous theorem.*

Now we consider the u-resultant approach, supposing resultant matrices $M(u), S$ are available, where S contains only scalars. Recall that the factors of the u-resultant are in bijective correspondence with the vectors representing the common roots of the input algebraic system. Given its coefficients in the monomial basis, the u-resultant can be factorized by standard methods (e.g., [Zip93]). This can be rather expensive, therefore we wish to rely only on its values.

Problem 7.4 *With no expansion in the monomial basis, express the factors of $R(u) = \det M(u)/\det S$, where $u = (u_0, \ldots, u_n)$.*

The primitive-element method of [Can88] expresses the factors of $R(u)$, and hence all affine roots, via the roots of a univariate polynomial and a set of n univariate rational expressions. The latter expressions, when specialized at the roots of the polynomial, yield the roots of the system. This means of expressing the roots implicitly can be particularly useful, for instance, if they have to be compared to other algebraic numbers or if we wish to store all roots in order to compute only a small number of them upon demand later on.

Theorem 7.5 *There is a Monte Carlo algorithm that solves Problem 7.4 by the primitive-element method (i.e., by computing n rational expressions and a univariate polynomial as above) with bit complexity $O^*(m^2n^2V_0 \log \|R(u_0)\|)$, where $\log \|R(u_0)\|$ bounds the bit length of the coefficients of the polynomials obtained by specializing the $u_1, \ldots, u_n$ in $R(u)$. This yields a description of all common roots of an algebraic system. The same results are achieved with a Las Vegas algorithm with bit complexity $O^*(m^2n^2V_0^2D\ell)$.*

Proof and Algorithm. The primitive-element algorithm of [Can88] (or the black-box method of [KT88]) reduces factoring to the computation and manipulation of $2n+1$ univariate polynomials in u_0, denoted by R^0, R_i^+, R_i^-, $i = 1, \ldots, n$. For details, see the proof of lemma 2.2 in [Can88]. Each new polynomial is defined by specializing the variables $u_1, \ldots, u_n$ to randomly selected constants, hence yielding univariate polynomials in u_0. These polynomials have degree V_0. Their output-sensitive coefficient length is in $O(\log \|R(u_0)\|)$. Another bound on this length is V_0 times the length of the coefficients in $R(u)$, hence $O(V_0D\ell)$.

We may compute any univariate polynomial via $V_0 + 1$ evaluations, each by applying the Monte Carlo version of theorem 5.3. The total bit complexity for expanding the $2n+1$ polynomials is $O^*(m^2n^2V_0q)$. Here q expresses the sum of the input and the output bit sizes which is asymptotically equal to the actual bit size of the coefficients of the univariate polynomials, expressed by $\log \|R(u_0)\|$.

The primitive-element algorithm computes the first subresultant of the square-free parts of $R_i^+(u_0)$ and $R_i^-(u_0)$, for $i = 1, \ldots, n$, and reduces the computed polynomials modulo $R^0(u_0)$. These steps have complexity dominated by the $O(n)$ univariate GCD computations, each with bit complexity in $O(V_0^2D\ell)$ if we use FFT. The total complexity of these steps is dominated by the cost of computing the $2n+1$ univariate polynomials. This procedure yields n univariate rational functions, whose specializations at the roots of $R^0(u_0)$ yield the original system's common zeros.

The Las Vegas version of this algorithm is obtained by applying the Las Vegas version of theorem 5.3. Then, in the above bounds, we let $q = V_0D\ell$ and the claim follows. □

Corollary 7.6 *There is a Las Vegas algorithm that solves Problem 7.4 by numerically approximating the factors of the u-resultant with bit complexity $O^*(m^2nV_0(m\beta + \log \|R(u_0)\|))$, where $\log \|R(u_0)\|$ is as above and β bounds the output precision of the factors' coefficients. This yields a Las Vegas algorithm for algebraic system solving with the same complexity, where β bounds the output precision of the roots' coordinates.*

Proof and Algorithm. We start with the primitive-element algorithm supporting theorem 7.5 which computes a univariate polynomial $R^0(u_0)$ and n univariate rational expressions in u_0. When the latter are specialized at the roots of $R^0(u_0)$, they yield the factors' coefficients, i.e., the system's common zeros. The most expensive step here is solving $R^0(u_0)$ by the Monte Carlo algorithm of theorem 7.2, in $O^*(m^2nV_0(q \log \beta + m\beta))$ bit operations. Recall that the additive factor $m\beta$ is due to the scaling of M in order to obtain the output with β bits. Moreover, the $\log \beta$ factor disappears from the final bound, as in the proof of theorem 7.5.

Now $q = \log \|R(u_0)\| + \beta$, and due to the $O^*(\cdot)$ notation, the bound stated at the claim follows. Observe that this complexity dominates that of the Monte Carlo algorithm supporting theorem 7.5.

The factors' coefficients give the system's roots, which can be checked by substitution into the given equations. The cost of this verification step is dominated, thus yielding a certified randomized method. □

References

[ABM99] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, pages 197–203, 1999.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.

[BCS97] P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory.* Springer, Berlin, 1997.

[BEPP99] H. Brönnimann, I.Z. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theoretical Computer Science, Special Issue on Real Numbers and Computers*, 210(1):173–197, 1999.

[Ber02] D.J. Bernstein. Multidigit arithmetic for mathematicians. *Adv. in Applied Math.*, 2002. To appear.

[BGY80] R.P. Brent, F.G. Gustavson, and D.Y.Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximations. *J. Algorithms*, 1:259–295, 1980.

[BP03] D. Bini and V.Y. Pan. *Polynomial and Matrix Computations*, volume 2: Selected Topics. Birkhäuser, Boston, 2003. To appear.

[Can88] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. ACM Symp. Theory of Computing*, pages 460–467, 1988.

[Can90] J. Canny. Generalised characteristic polynomials. *J. Symbolic Computation*, 9:241–250, 1990.

[CKL89] J.F. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In *Proc. ACM Intern. Symp. on Symbolic & Algebraic Comput.*, pages 121–128, 1989.

[CLO98] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Number 185 in Graduate Texts in Mathematics. Springer-Verlag, New York, 1998.

[D'A02] C. D'Andrea. Macaulay-style formulas for the sparse resultant. *Trans. of the AMS*, 354:2595–2629, 2002.

[DE01] C. D'Andrea and I.Z. Emiris. Computing sparse projection operators. In *Symbolic Computation: Solving Equations in Algebra, Geometry, and Engineering*, volume 286 of *Contemporary Mathematics*, pages 121–139, Providence, Rhode Island, 2001. AMS.

[EC95] I.Z. Emiris and J.F. Canny. Efficient incremental algorithms for the sparse resultant and the mixed volume. *J. Symbolic Computation*, 20(2):117–149, 1995.

[EGV00] W. Eberly, M. Giesbrecht, and G. Villard. Computing the determinant and Smith form of an integer matrix. In *Proc. Annual IEEE Symp. Foundations of Comp. Sci.*, pages 675–685. IEEE Computer Society Press, Los Alamitos, Calif., 2000.

[Emi98] I.Z. Emiris. A complete implementation for computing general dimensional convex hulls. *Intern. J. Computational Geometry & Applications, Special Issue on Geometric Software*, 8(2):223–253, 1998.

[EP02] I.Z. Emiris and V.Y. Pan. Symbolic and numeric methods for exploiting structure in constructing resultant matrices. *J. Symbolic Computation*, 33:393–413, 2002.

[Kal02] E. Kaltofen. An output-sensitive variant of the baby steps/giant steps determinant algorithm. In *Proc. ACM Intern. Symp. on Symbolic & Algebraic Comput.*, pages 138–144. ACM Press, New York, 2002.

[KP91] E. Kaltofen and V.Y. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. 3rd Ann. ACM Symp. on Parallel Algorithms and Architectures*, pages 180–191. ACM Press, New York, 1991.

[KS91] E. Kaltofen and B.D. Saunders. On Wiedemann's method for solving sparse linear systems. In H.F. Mattson, T.Mora, and T.R.N. Rao, editors, *Proc. Intern. Symp. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 539 of *Lect. Notes in Comp. Science*, pages 29–38, Berlin, 1991. Springer-Verlag.

[KT88] E. Kaltofen and B.M. Trager. Computing with polynomials given black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 296–305. IEEE Comp. Society Press, Los Alamitos, Calif., 1988.

[KV01] E. Kaltofen and G. Villard. On the complexity of computing determinants. In K. Shirayanagi and K. Yokoyama, editors, *Proc. Fifth Asian Symp. Computer Math.*, volume 9 of *Lect. Notes in Computing*, pages 13–27. World Scientific, Singapore, 2001.

[Man93] D. Manocha. Efficient algorithms for multipolynomial resultants. *The Computer Journal*, 36(5):485–496, 1993.

[MC92] D. Manocha and J. Canny. The implicit representation of rational parametric surfaces. *J. Symb. Comput.*, 13:485–510, 1992.

[MC93] D. Manocha and J. Canny. Multipolynomial resultant algorithms. *J. Symbolic Computation*, 15(2):99–122, 1993.

[McN93] J.M. McNamee. A bibliography on roots of polynomials. *J. Computat. Applied Math.*, 47:391–394, 1993.

[Mou98] B. Mourrain. Computing isolated roots by matrix methods. *J. Symbolic Computation*, 26(6):715–738, 1998. Special Issue on Symbolic-Numeric Algebra for Polynomials.

[Pan92] V.Y. Pan. Parametrization of Newton's iteration for computations with structured matrices and applications. *Comp. & Math. (with Appl.)*, 24(3):61–75, 1992.

[Pan97] V.Y. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Rev.*, 39(2):187–220, 1997.

[Pan01] V.Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms.* Birkhäuser / Springer, Boston / New York, 2001.

[Pan02a] V.Y. Pan. Randomized acceleration of fundamental matrix computations. In *Proc. STACS*, LNCS, pages 215–226. Springer-Verlag, Berlin, 2002.

[Pan02b] V.Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *J. Symbolic Computation*, 33(5):701–733, 2002.

[PY01] V.Y. Pan and Y. Yu. Certification of numerical computation of the sign of the determinant of a matrix. *Algorithmica*, 30:708–724, 2001.

[Sto02] A. Storjohann. High-order lifting. In *Proc. ACM Intern. Symp. on Symbolic & Algebraic Comput.*, pages 246–254. ACM Press, New York, 2002.

[Too63] A.L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Math. Doklady*, 3:714–716, 1963.

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

[Wie86] D.H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986.

[Zip93] R. Zippel. *Effective Polynomial Computation.* Kluwer Academic Publishers, Boston, 1993.

Appendix

Integer determinants by the Kaltofen-Villard algorithm

The main stages of the algorithm in [KV01, sect.3], supporting theorem 2.1, are:
1. Precondition M.
2. Compute $B^{[i]} := X^T M^i Y$ for $i = 0, 1, \ldots, \Theta(m/t)$, where X, Y are random $m \times t$ matrices, $t \in \mathbb{Z}$ is chosen in $[1, m]$; this is achieved with a baby step / giant step technique. The most costly operations in this stage is substage 2.2, of computing a power M^r of M for $1 \le r \le t$, and substage 2.3, of computing the Krylov sequence $X^T M^{rk}, k = 1, 2, \ldots, s$ for $s = O(m/(tr))$.
3. Compute the minimal matrix generator for the sequence $(B^{[i]})_{i \ge 0}$; this reduces to finding t linearly independent vectors in the kernel of the block Toeplitz matrix $T := (B^{[i]}), i = 0, \ldots, \lceil 2m/t \rceil + 3$.
4. Compute $\det M$, which is equal to the ratio of the leading and trailing coefficients of the determinant of this generator, with a high probability.

For computations at stage 3 without fast matrix multiplication, [KV01] only states the bound $O^*(m^2 t^3 L)$ for $k = 0$, in the notation of section 2, and derives this bound by using Levinson-Durbin's algorithm. The resulting overall bit cost estimate $O^*(m^{10/3} L)$ was stated and derived in [KV01]. In fact one may balance the bit cost at all stages properly to derive here even a slightly better bound $O^*(m^{23/7} L)$.

The bound (2) implies $O(m^2 t^2 L)$ for $k = 0$ which can be achieved in at least three ways. According to [Pan02a], we may interchange the matrix rows and columns of T in order to bound the displacement rank by t, then apply the Morf-Bitmead-Anderson (MBA) divide-and-conquer algorithm [Pan01, ch.5], complemented by compression of displacement generators by Pan [Pan92, Appendix], and by the randomized preconditioning of Kaltofen and Saunders [KS91]. Alternatively, one may apply the block MBA algorithm to the original block Toeplitz matrix T, to yield the same complexity bound increasing the number of random parameters involved.

Thirdly, one may apply the block version of the algorithm of Brent, Gustavson and Yun [BGY80] instead of the block MBA algorithm. The latter recipe, in the equivalent form of using Knuth-Schönhage's block half-gcd algorithm (also supporting the same complexity bound at stage 3 and consequently the overall bound $O^*(m^{16/5} L)$) was cited in [KV01], although only in conjunction with the algorithms using fast matrix multiplication (for $k > 0$ this is the multivariate block half-gcd algorithm). So the bound $O^*(m^{16/5} L)$ is implicit in [KV01]. All the cited bounds rely on employing the CRA.

The extension to the multivariate case is the simplest with the first approach, exploiting the displacement rank, because the computations can be performed modulo the powers $k_i + 1$ of the variables x_i whose degrees are k_i in the output. This is prohibitive in the third approach with the block half-gcd algorithm, which does not apply to matrix polynomials. Then, shifting to Toom's evaluation-interpolation over the integers is imperative, with more tedious bit complexity estimates.

Application of Tellegen's theorem

The rest of the Appendix discusses the application of Tellegen's theorem. The matrix-vector multiplication algorithm of [EP02] has arithmetic complexity $O^*(mn)$ under the RAM model. Since it has no branching, and all computed quantities are linear combinations of the vector entries (see below), the same complexity holds under the linear complexity model, which is a refinement of SLPs [BCS97, ch.13]. Hence we can apply Tellegen's theorem, to obtain the same complexity bound for vector multiplication with the transposed matrix. The latter implies the same bound under the RAM model.

It remains to show that the matrix-vector multiplication of [EP02, alg.4.5] has no branching and all computed quantities are linear in the vector entries; we call *valid* any operation that satisfies these two conditions. We shall use the notation of [EP02] and study the steps of its algorithm. In particular, the vector contains the coefficients of polynomial g, whereas the matrix entries are coefficients of polynomial f. Steps 1 and 2 evaluate monomials, involving primes and the support point coordinates, which we assume are in the base field, denoted by K; hence they are valid.

Let us now examine step 3, composed of the following substeps: (a) Computing $V^T V$ is independent of the vector and reduces to solving a triangular Toeplitz system, by the remark of [EP02], hence it is valid. (b) Solving the Vandermonde system defined by matrix V is tantamount to interpolation, an operation composed of three stages: (b.1) Use a fan-in scheme to compute $L(x) = \prod (x - v_i)$, $L'(x)$, where v_i are the interpolation points, defined by V. This stays in K and uses no branching, hence it is valid. (b.2) Evaluating $L'(v_i)$ for a fixed set of values of v_i is

valid. (b.3) Computing $L(x)\sum_i c_i/(L'(v_i)(x-v_i))$ is valid, since the computation either stays within K, when the c_i are coefficients of polynomial f, or is linear in the c_i entries of the given vector, when dealing with polynomial g. (c) Compute the product of Hankel matrix V^TV with vector c'. This is valid by [BCS97, cor.13.13], even when the entries of c' are linear in the coefficients of g, since they remain linear in these coefficients. (d) Multiply the corresponding values of f, g to obtain the values of their product; this is again valid.

Step 4 is the solution of a transposed Vandermonde system, and has 3 substeps by [Pan01, thm.3.4.1,eqt.(3.4.3)]. (a) The computation of an expression $\prod(x-v_i)$, independent of the given vector entries. It requires no branching, as discussed above. In equation (3.4.3) of [Pan01, p.81], we can choose $f=0$ or $f=1$, leading to a circulant matrix Z_f, which is independent of the vector entries and can be computed without branching. (b) One must compute the matrix-vector product between matrix Z_f and the input vector, then multiply the product vector with the reversion matrix J and a Vandermonde matrix V. All of these operations are valid by [BCS97, cor.13.13]. (c) Lastly, the algorithm computes the inverse of a diagonal matrix, which is independent of the input vector and can be determined without branching. Hence, step 4 is valid.