The structure of sparse resultant matrices

Ioannis Z. Emiris
INRIA, B.P. 93, Sophia-Antipolis 06902, France.
emiris@sophia.inria.fr
http://www.inria.fr/safir/whoswho/emiris

Victor Y. Pan
Mathematics and Computer Science Department, CUNY, Bronx, NY 10566, USA.
vpan@lcvax.lehman.cuny.edu

Abstract

Resultants characterize the existence of roots of systems of multivariate nonlinear polynomial equations, while their matrices reduce the computation of all common zeros to a problem in linear algebra. Sparse elimination theory has introduced the sparse resultant, which takes into account the sparse structure of the polynomials. The construction of sparse resultant, or Newton, matrices is a critical step in the computation of the resultant and the solution of the system. We exploit the matrix structure and decrease the time complexity of constructing such matrices to roughly quadratic in the matrix dimension, whereas the previous methods had cubic complexity. The space complexity is also decreased by one order of magnitude. These results imply similar improvements in the complexity of computing the resultant itself and of solving zero-dimensional systems. We apply some novel techniques for determining the rank of rectangular matrices by an exact or numerical computation. Finally, we improve the existing complexity for polynomial multiplication under our model of sparseness, offering bounds linear in the number of variables and the number of nonzero terms.

1 Introduction

Resultants characterize the solvability of zero-dimensional systems of multivariate nonlinear polynomial equations, and their matrix formulae reduce the computation of all common solutions to a matrix eigenproblem. Resultants have a long and rich history in the context of classical elimination. More recently, sparse elimination theory introduced the *sparse resultant*, which generalizes the classical resultant and whose degree depends on the monomial structure of the polynomials, thus leading to tighter bounds and faster algorithms for systems encountered in application areas; section 5 gives a formal introduction. Sparse resultant matrices, also known as *Newton matrices,* generalize Sylvester and Macaulay matrices, and from their determinant the sparse resultant can be computed. This paper identifies and exploits the structure of Newton matrices, thus deriving better time and space complexity bounds for their construction, the computation of the sparse resultant and the solution of polynomial systems.

Construction and manipulation of Newton matrices is a critical operation in some of the most efficient known algorithms for solving zero-dimensional systems of equations [3, 12, 5, 13]. Our practical motivation is the real-time solution of systems with, say, up to 10 variables; or the computation of the resultant polynomial, for instance in graphics and modeling applications where the implicit expression of a curve or surface is precisely the resultant. Such systems may give rise to matrices with dimension in the hundreds or even higher, as illustrated by specific examples in table 2. By palliating the effects of matrix size, our work deals with what is probably the Achille's heel of Newton's matrices, in comparison to the Bézout/Dixon matrix, which is typically smaller.

The main contribution of the present paper is to construct Newton matrices in time complexity quasi-quadratic in the matrix dimension, which constitutes an improvement of one order of magnitude. The same improvement for space complexity yields a quasi-linear bound. Analogous improvements are then obtained for computing the sparse resultant and, eventually, for solving systems of polynomial equations. These bounds ultimately rely on the Fast Fourier Transform (FFT). Yet, other polynomial multiplication methods, such as Karatsuba's, may offer simpler though asymptotically slower alternatives; the latter may be advantageous in certain circumstances, as discussed in section 8. Table 1 compares the existing and the achieved complexities, in terms of matrix row and column dimension, respectively denoted a and c and the number of variables n, as explained in section 6. Note that $a > c$ and typically $a, c \gg n$.

Table 1: Asymptotic complexity for matrix construction

method	time	space
previous	a^2c	ac
Karatsuba	$a^{\lg 3}cn$	cn
FFT	c^2n	cn

There are certain auxiliary results of independent interest. The reduction of vector-by-matrix multiplication to polynomial multiplication calls for efficient sparse polynomial evaluation and interpolation. The proposed algorithms have *linear* time and space complexity in terms of n and the cardinality of the supports, i.e., the sets of nonzero terms. This improves the known algorithms for polynomial evaluation and interpolation under our model of sparseness.

Furthermore, we sketch fast algorithms for computing the rank of a rectangular matrix. We refer to an exact-

arithmetic randomized method as well as a numerical approximative method, both exploiting matrix structure, in particular, fast matrix-vector multiplication.

It shall become clear that our results make use of a variety of techniques for polynomial arithmetic and structured matrix computation. This allows us to improve the existing straightforward bounds, thus contributing to the development of resultant-based methods, a field of active research. The most relevant work is that on Macaulay matrices [3], which our results generalize, and the related and alternative viewpoint adopted in [13].

This paper is organized as follows. The next section expands on related work. Section 3 indicates two efficient rank determination algorithms, and section 4 considers efficient sparse polynomial arithmetic. Certain important properties of the Newton matrix are investigated in section 5, including its premultiplication by a vector. Section 6 improves the complexity of a known algorithm for constructing such matrices by exploiting their structure. Computing the sparse resultant itself is investigated in section 7. We conclude with extensions of our results in section 8.

2 Related work

Resultant-based approaches to studying and solving systems of polynomial equations have a long history, a more comprehensive account of which is in [8]. This article also includes several important references to other related work that had to be omitted here because of space restrictions.

Recent interest in matrix-based methods is supported by certain practical results that have established resultants, along with Gröbner bases and continuation techniques, as the method of choice in solving zero-dimensional polynomial systems [17, 3, 12, 5, 13]. A generalization of the classical resultant was introduced in the context of sparse elimination theory (outlined in section 5). Two main algorithms, generalizing Sylvester's as well as Macaulay's constructions, have been proposed for constructing Newton, or sparse resultant, matrices: The subdivision-based algorithm of [2] (subsequently improved and generalized in [4, 16]) and the incremental algorithm of [7], which constructs a rectangular matrix and then obtains a square nonsingular submatrix.

Canny, Kaltofen and Lakshman [3] studied the structure of Macaulay matrices Our results generalize their approach. Independently, Mourrain and Pan [13] formalized the Toeplitz- or Hankel-like structure of general resultant matrices, including Macaulay, Bézout and Newton matrices. Their work provides a related and alternative viewpoint to our approach.

An auxiliary issue (also important on its own right) is to devise algorithms for multiplying sparse multivariate polynomials within the computational complexity bounds expressed via the support cardinality or the Newton polytope. The existing general bounds are interesting only in the dense case [1]. Sparse interpolation has received a lot of attention; see the algorithms in [19, 9], supporting complexity linear in the product of n, the maximum degree in any single variable and a bound on the number of monomials. Section 4 improves these bounds by exploiting the structure of nonzero terms, and generalizes [3] from completely dense supports to arbitrary supports.

3 Fast rank tests

We describe an exact-arithmetic probabilistic method, as well as a rational approximative method for testing whether a rectangular matrix has full rank. These results are important for improving the known algorithms for solving polynomial systems of equations as well as of independent interest. Here we offer an outline of the main properties and refer the reader to [8] for further discussion, including the description of the algorithms and all proofs.

All time complexity bounds are in terms of *arithmetic complexity;* hereafter "ops" stands for "arithmetic operations." Space complexity includes the input and output storage, unless when we explicitly refer to "additional" storage space. We let $O^*(c)$ stand for $O(c\log^v c)$ for any fixed constant v independent of c.

Let W^T denote the transpose of a matrix or of a vector W, I_k denote the $k \times k$ identity matrix, O the rectangular null matrix of an appropriate size, and $[A, B]$ the 1×2 block matrix with A and B as its blocks. This paper makes heavy use of dense structured matrices; for a comprehensive account of their definitions and properties, the reader may consult [1]. We recall that a $k \times k$ Toeplitz matrix can be multiplied by a vector in $O(k \log k)$ ops [1, sect. 2.5] and in $O(k)$ storage space, based on the FFT.

Lemma 3.1 [15, fact 7.2] *Let M be an $a \times c$ matrix of a rank r with entries in any field. Let L and U^T denote two unit lower triangular Toeplitz matrices of sizes $a \times a$ and $c \times c$, respectively, defined by the $a-1$ and $c-1$ entries in each of the matrices' first column. Suppose that these $a+c-2$ entries are random, chosen independently of each other from a fixed finite subset T of the base field or of its extension, assuming the uniform probability distribution on T. Then the $r \times r$ trailing principal submatrix of the matrix LMU is nonsingular with a probability at least $1 - 2r/|T|$, which increases at least to $1 - r/|T|$ if $r = \min(a, c)$.*

Theorem 3.2 [18, thm. 1], [10, lem. 2] *The determinant of a $c \times c$ matrix W (defined over any field of constants having a finite subset T of at least $|T| \geq 50c^2 \log_2 c$ elements) can be computed in $O(c)$ multiplications of $O(c)$ vectors by W and, in addition, in $O^*(c^2)$ ops requiring $O(c)$ additional storage space, by means of a randomized algorithm, using c random parameters and having a failure probability of at most $2c/|T|$.*

Theorem 3.3 *Let C and G be the time and space complexities, respectively, of multiplying an $1 \times a$ vector by an $a \times c$ matrix M, with $a \geq c$, over any field of constants having at least $50c^2 \log_2 c$ elements. Then there exists a randomized algorithm that tests whether M has full rank in $O(cC + ac \log c)$ ops using $O(G + a)$ storage space, not including the cost of generating the $a + c - 1$ required random parameters.*

Proof We apply the construction of lemma 3.1 and then the algorithm implied by theorem 3.2. The full algorithm is detailed in [8]. □

Over the complex field **C** or its subfields, we may test by a *floating point* computation whether matrix M has full rank, that is, whether M has c singular values whose moduli exceed a fixed small positive tolerance value ϵ. This can be achieved by means of any black box algorithm for computing the Singular Value Decomposition (SVD) of M. A much less costly algorithm of [14] exploits the structure of M. The algorithm avoids computing SVD and is *rational*, that is, only involves field operations, which can also be performed in exact arithmetic.

Theorem 3.4 *Let C and G be the time and space complexities, respectively, of multiplying an $1 \times a$ vector by a complex $a \times c$ matrix M, with $a \geq c$. Then there exists a rational randomized algorithm that tests numerically, in $O(aC + a^2)$ ops and $O(G + a)$ storage, whether M has full rank within a given tolerance $\epsilon > 0$.*

Proof The algorithm applies Lanczos' algorithm [1, alg. 2.3.1] to reduce M to a tridiagonal form T, then computes the sequence of the signs of $\det(I_i - \epsilon^2 T_i)$, for $i = 1, \ldots, a$, where T_i is the $i \times i$ leading principal submatrix of T. This sign sequence determines the rank within ϵ. The full algorithm is detailed in [14, 8]. □

Corollary 3.5 *The algorithm of theorem 3.3 yields as by-product the determinant of a square matrix and can be modified in a straightforward way to yield an algorithm that finds the rank of a rectangular matrix in $O(cC \log c + ac \log^2 c)$ ops and $O(G + a)$ storage. The algorithm of theorem 3.4 computes the rank of a rectangular matrix as a by-product. For both algorithms, the input rectangular matrix can be rank deficient.*

4 Support evaluation and sparse polynomial multiplication

We examine polynomial multiplication in the setting of sparse elimination theory, where polynomials are defined by their supports. Certain results of this section improve upon the corresponding results in [8]; the latter article also contains the proofs and algorithms that are omitted here.

We will work in the ring of Laurent polynomials $P = K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$, where K is any given field of characteristic zero. The *support* of $f \in P$ is a subset of $\mathbf{Z}^n$ denoted $\mathrm{supp}(f)$ and containing all the exponent vectors of monomials with nonzero coefficients in f. If $S = \mathrm{supp}(f) \subset \mathbf{Z}^n$, then

$$f = \sum_{a \in S} c_a x^a, \qquad x^a = \prod_{i=1}^{n} x_i^{a_i},$$

where $a = (a_1, \ldots, a_n) \in \mathbf{Z}^n$, $c_a \in K$. In dealing with supports, we slightly abuse terminology and speak of a monomial in a support, referring to the monomial defined by the integer point representing its exponent. In the sequel, we assume, without loss of generality, that all polynomial supports contain the origin; this can be achieved by a translation of the supports.

For every polynomial, there is an associated *Newton polytope*, which is the convex hull of the support. Newton polytope generalizes the classical notion of total degree of an n-variate polynomial; for a completely dense polynomial, the Newton polytope is the n-dimensional unit simplex. Define the *Minkowski sum* $A + B$ of two point sets A and B in $\mathbf{R}^n$ as the point set $A + B = \{a + b \mid a \in A, b \in B\}$. If A, B are convex polytopes, then so is $A + B$. For further information on sparse elimination see [16, 7, 5] and their references.

The following algorithms and their complexity analysis are of independent interest as they demonstrate that the complexity of polynomial multiplication, evaluation and interpolation on some special sets of points depends on the corresponding support cardinalities and Newton polytope volumes; these two are asymptotically equivalent. This discussion complements the known results on sparse evaluation and interpolation by settling the case where sparseness is measured by the support.

To simplify the notation, we assume when we discuss evaluation that all monomials have non-negative exponents.

Lemma 4.1 *Consider a set S of s positive integers, such that $S \subset \mathbf{N} \cap [0, d]$, for some positive integer d. If we are given a value p, we may evaluate all powers of p with exponents in S by using $O^*(s + \sqrt{d})$ ops and $O(s)$ storage space.*

Lemma 4.2 *Consider a set S of s monomials in n variables, such that the exponent of every monomial in the ith variable lies in $[0, d]$, for $i = 1, 2, \ldots, n$. Given n scalar values $p_1, p_2, \ldots, p_n$, one may evaluate all the monomials of S at these values in $O^*(sn + n\sqrt{d})$ ops and $O(sn)$ space by the corresponding algorithm of [8].*

Storage space can be reduced to $O(s)$ under the hypothesis that the exponent vectors representing monomials have "short" entries so that they can be stored in constant space.

The following multiplication algorithm extends the approach of [3, sect. 3], based on the widely used evaluation-interpolation scheme, with node sets from a special customary class, also used in [9, 19, 1]. We will focus on multiplication, but our algorithm improves sparse evaluation and interpolation as by-product.

Algorithm 4.3 (Sparse polynomial multiplication)
Input: n-variate polynomials $f, g \in P$ with supports $A, B \subset \mathbf{Z}^n$, respectively. Also given is a set of points $S \subset \mathbf{Z}^n$ such that $A + B \subset S$, so that S contains the support of fg.
Output: The product fg.

Computations:

1. Let $A = \{a_1, \ldots, a_{|A|}\}$, with each $a_k \in A$ written as $(a_{k1}, \ldots, a_{kn})$. Let $S = \{m_1, \ldots, m_s\} \subset \mathbf{Z}^n$, where s is the set cardinality. Pick n distinct primes $p_1, \cdots, p_n$, supposed to be readily available.

2. Compute the values $v_k = \prod_{i=1}^{n} p_i^{a_{ki}}$ of the monomials in A at the point $(p_1, \ldots, p_n)$, for $k = 1, 2, \ldots, |A|$. Note that $v_k^j = \prod_{i=1}^{n} (p_i^j)^{a_{ki}}$ is the value of x^{a_k} at $(p_1^j, \ldots, p_n^j)$. Therefore, multiplication of the row vector of the coefficients of f by the $|A| \times s$ matrix

$$\begin{bmatrix} 1 & v_1 & \cdots & v_1^{s-1} \\ \vdots & \vdots & & \vdots \\ 1 & v_{|A|} & \cdots & v_{|A|}^{s-1} \end{bmatrix}$$

   expresses the evaluation of f at the points $(p_1^j, \cdots, p_n^j)$ for $j = 0, 1, \cdots, s-1$. We append rows of powers $1, v_i, \cdots, v_i^{s-1}$, for distinct v_i, $i = |A|+1, \cdots, s$, to the matrix above in order to obtain an $s \times s$ Vandermonde matrix V.

3. Let c_f be the $s \times 1$ column vector whose first $|A|$ entries are the coefficients of f, in the order defined by an arbitrary but fixed monomial sequence $(a_1, \ldots, a_{|A|})$; let the last $s - |A|$ entries be zeros. Then the column vector of the values of f at $v_1, \ldots, v_{|A|}$ is expressed as $V^T c_f = (V^T V)(V^{-1} c_f)$. Compute $V^T V$, $V^{-1} c_f$ and their product as discussed in [3, sect. 3a] or [1, ch. 2]. Analogously evaluate the polynomial g at the same points. Then multiply the values of f and g pointwise, thus computing the values of fg at every point $(p_1^j, \ldots, p_n^j)$, $j = 0, \ldots, s-1$.

4. Let l_{fg} and c_{fg} denote the vectors of the product values and of the unknown coefficients fg ordered, respectively, by $(p_1^j, \ldots, p_n^j)$ for $j = 0, \ldots, s-1$ and by

a fixed monomial sequence $(m_1, \ldots, m_s)$. Compute w_i as the value of m_i at $(p_1, \ldots, p_n)$ and let W be the $s \times s$ Vandermonde matrix $[w_i^{j-1}]$, analogous to V in step 2. Solve the transposed Vandermonde system $W^T c_{fg} = l_{fg}$, e.g. by applying the algorithm of [9]. The solution c_{fg} defines fg; some coefficients are zero if and only if the support of the product is a proper subset of S.

Theorem 4.4 *Given n-variate polynomials $f, g \in P$ with supports $A, B \subset \mathbf{Z}^n$, respectively, and given a point set $S \subset \mathbf{Z}^n$ such that $A + B \subset S$, the product fg can be computed by algorithm 4.3 by using $O^*(sn + n\sqrt{d})$ ops and $O(sn)$ space, where d is the maximum degree of each input polynomial in any variable and $s = |S|$.*

Proof In steps 2 and 4, the Vandermonde matrices are defined by at most s values v_i and w_i, respectively. These are the values of all monomials from the set A at a point $(p_1, \cdots, p_2)$; according to lemma 4.2, such values can be computed in $O^*(sn+n\sqrt{d})$ ops since the maximum degree of the product polynomial in each variable is bounded by $2d$. For interpolation, note that all monomials in fg belong to S. The storage requirement is $O(sn)$. Steps 3 and 4 take each $O(s \log^2 s)$ ops and $O(s)$ storage space, due to the techniques of [9, 3] (also see [1, sect. 2.6]). Both estimates exploit the structure of the Hankel matrix $V^T V$. □

Remark 4.5 *The entries of $V^T V$ equal the power sums of some polynomial roots. These entries are computed via the identities involving the symmetric functions of the corresponding coefficients, by solving a Toeplitz linear system of equations. In fact, this Toeplitz linear system is* triangular, *so its solution is substantially simpler than that stated in [3, sect. 3a].*

With the notation of the previous theorem, let f_i, g_i be polynomials in P, $i = 1, \ldots, k$, and $A_i, B_i \subset \mathbf{Z}^n$ be the respective supports. If $S \subset \mathbf{Z}^n$ is given so that $\cup_i (A_i + B_i) \subseteq S$, then computing $\sum_{i=1}^k f_i g_i$ has time complexity $O^*(s\,n\,k + nk\sqrt{d})$ and space complexity $O(sn)$. This straightforward corollary can be improved when k is bounded with respect to n, by the approach of [13, prop. 24]. Moreover, by choosing a special set of points, the following algorithm and theorem arrive at a small improvement upon [13, prop. 24].

We modify algorithm 4.3 (and use its notation), in order to compute part of product FG, where

$$F = \sum_{i=1}^{n+1} x_{n+1}^{n-1+i} f_i, \; G = \sum_{i=1}^{n+1} x_{n+1}^{n+1-i} g_i$$

are $(n+1)$-variate polynomials and x_{n+1} is a new variable. If we consider FG as a polynomial in x_{n+1}, the coefficient of x_{n+1}^{2n} equals $\sum_{i=1}^{n+1} f_i g_i$.

Algorithm 4.6 (Sum of polynomial products)
Input: n-variate polynomials $f_i, g_i \in P$ $i = 1, \ldots, n+1$, with respective supports $A_i, B_i \subset \mathbf{Z}^n$. $S \subset \mathbf{Z}^n$ is given, such that $\cup_i (A_i + B_i) \subseteq S$.
Output: $\sum_{i=1}^{n+1} f_i g_i$.

Computations:

1. Pick primes $p_1, \ldots, p_n, p_{n+1}$ and compute the values $w_k = \prod_{i=1}^n p_i^{m_{ki}}$, $k = 1, \ldots, s$.
2. Let $A_i' = \{(a, n-1+i) : a \in A_i\} \subset \mathbf{Z}^{n+1}$. Evaluate all A_i' monomials, for $i = 1, \ldots, n+1$, by multiplying the appropriate w_k by p_{n+1}^{n-1+i}. The new values define Vandermonde matrix V_F expressing evaluation of F. Construct the coefficient vector c_F and compute the evaluation vector $l_F = V_F^T c_F$. Analogously proceed for polynomial G. Then multiply pointwise the two evaluation vectors in order to obtain l_{FG}.
3. Let l be the leading s-subvector of l_{FG}, after dividing the jth entry, $j = 0, \ldots, s-1$, by $(p_{n+1}^{2n})^j$. Let W be the $s \times s$ Vandermonde matrix defined by the w_k and solve $W^T c = l$ for the coefficient vector c of $\sum_{i=1}^{n+1} f_i g_i$.

Theorem 4.7 *Consider polynomials $f_i, g_i \in P$, $i = 1, \ldots, n+1$. Let $A_i, B_i \subset \mathbf{Z}^n$ be the respective supports and let $S \subset \mathbf{Z}^n$ be such that $\cup_i (A_i + B_i) \subseteq S$. Then computing $\sum_{i=1}^{n+1} f_i g_i$ by algorithm 4.6 has time complexity $O^*(sn + n\sqrt{d})$ and space complexity $O(sn)$, where d is the maximum degree of f_i, g_i in any variable.*

Proof Step 1 is within the complexity bounds by lemma 4.2. Step 2 takes $O^*(s)$ per polynomial and total space $O(sn)$ to compute all necessary values. Matrix V_F has dimension equal to the cardinality of $\mathrm{supp}(F)$, which is at most sn because $\mathrm{supp}(F) = \cup_i A_i'$ and $A_i \subset S$. Hence, computing l_F and, subsequently, l_G and l_{FG}, all have complexity $O^*(sn)$ by the proof of theorem 4.4. Step 3 has complexity $O^*(s)$. □

Observe that the computation of $l_F = V_F^T c_F$ can be simplified to computing $[I_s, 0] V_F^T c_F$, since only a leading s-subvector of l_F is needed.

This theorem is instrumental in accelerating multiplication of a vector by sparse resultant matrix M constructed by our algorithms. S can be taken to be precisely $\cup_i (A_i + B_i)$. Typically, the exact computation of support S is expensive, so we can bound it by the integer lattice points lying in the Minkowski sum of the Newton polytopes of the f_i. In the dense context, $s = |S|$ was bounded simply as a function of the degrees, thus yielding a quite loose bound.

5 The Newton matrix and its premultiplication by a vector

In this section, we first recall some major concepts and results of sparse elimination theory and briefly describe the general problem of constructing Newton matrices, which express sparse resultants; we refer the reader to [16, 7, 5] and their references for a comprehensive presentation. We specify a simple randomized transformation of a rectangular Newton matrix into a square one, then show how to premultiply Newton matrices by vectors fast.

Consider a well-constrained polynomial system $f_1, \ldots, f_n \in P$. Given convex polytopes $A_1, \ldots, A_n \subset \mathbf{R}^n$, there is a real-valued function $MV(A_1, \ldots, A_n)$, called the *mixed volume* of $A_1, \ldots, A_n$. See [7] for a number of equivalent definitions of mixed volume and an efficient algorithm for its computation. Bernstein's theorem states that the mixed volume of the Newton polytopes associated to the polynomial system of equations $f_1 = \cdots = f_n = 0$ bounds the number of isolated roots of this system in $(\overline{K}^*)^n = (\overline{K} \setminus \{0\})^n$, where $\overline{K}$ is the algebraic closure of the base field. The mixed volume is typically much less than Bézout's bound for sparse polynomial systems. We recall that Bézout's bound on the number of (projective) roots is $\prod_i d_i$, where d_i is the total degree of the polynomial f_i, for $1 \leq i \leq n$.

Now we pass to the context of overconstrained systems $f_1, \ldots, f_{n+1} \in P$. The *sparse resultant* R of polynomials $f_1, \ldots, f_{n+1}$ is an irreducible polynomial in the f_i coefficients, which provides a necessary condition for solvability of the overconstrained system $f_1 = \cdots = f_{n+1} = 0$ over $(\overline{K}^*)^n$, i.e., it vanishes whenever there exists a solution in $(\overline{K}^*)^n$. R is a homogeneous polynomial in the coefficients of each f_i whose degree, denoted $\deg_{f_i} R$, is given by the following mixed volume.

$$\deg_{f_i} R = MV(f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_{n+1}). \qquad (1)$$

The algorithmic problem of computing R is typically solved by constructing square matrices, called *resultant matrices*, whose determinant is ideally R or, more generally, a nontrivial multiple of R. Even in the second case, the resultant matrix suffices for reducing the computation of all roots of $f_1 = \cdots = f_{n+1} = 0$ to a problem in linear algebra; see, for instance, [5].

For a nonempty set of monomials $S \subset \mathbf{Z}^n$, let

$$P(S) = \{f \in P \,:\, \mathrm{supp}(f) \subset S\} \subset P$$

be the vector space over some monomial basis in S, of dimension equal to the cardinality $s = |S|$. Hence a polynomial is represented by a vector, and a list of polynomials by a concatenation of vectors.

Definition 5.1 *Let $n+1$ polynomials $f_1, \ldots, f_{n+1} \in P$ have supports $A_1, \ldots, A_{n+1} \subset \mathbf{Z}^n$. Let $B_1, \ldots, B_{n+1} \subset \mathbf{Z}^n$ be the supports of polynomials $g_1, \ldots, g_{n+1} \in P$ such that the linear transformation*

$$\mu: \quad P(B_1) \times \cdots \times P(B_{n+1}) \to P\left(\textstyle\bigcup_{i=1}^{n+1}(A_i + B_i)\right), \quad (2)$$
$$[g_1, \ldots, g_{n+1}] \mapsto [g_1, \ldots, g_{n+1}]M = \left[\textstyle\sum_{i=1}^{n+1} g_i f_i\right]$$

is surjective for generic coefficients of the f_i and, moreover, the dimension of the domain of μ is at least as large as the dimension of the range, in other words, $\sum_i |B_i| \geq |\cup_i (A_i + B_i)|$. Then the matrix M of this transformation has entries in K and has at least as many rows as columns. This is a sparse resultant, *or* Newton, matrix *for the system $f_1, \ldots, f_{n+1}$.*

Observe that the coefficients of g_i in this definition are immaterial. Each entry of M is either zero or equal to a coefficient of some f_i. The rows of M are indexed by the points in B_i, so that the row corresponding to $b \in B_i$ expresses the polynomial $x^b f_i$. The columns of M are indexed by the points in $\cup_i (A_i + B_i)$, which is precisely the support of $\sum_i g_i f_i$. In short, Newton matrices are constructed in the same way as Sylvester and Macaulay matrices [17].

However, keeping with the philosophy of this paper, we store a Newton matrix by storing only the B_i and f_i, $i = 1, \ldots, n+1$, hence using $O(cn)$ space, where c denotes the number of matrix columns and bounds the cardinality of any B_i. This space bound relies on the hypothesis that a multi-index representing an integer exponent vector or, equivalently, a monomial takes $O(1)$ space. This is assumed in the rest of the paper and is justified by the observation that, typically, the list of n maximum degrees in any variable (denoted d) can be stored in constant space. The following well known theorem is the basis for computing nontrivial multiples of the resultant [17, 2, 8].

Theorem 5.2 *Consider any maximal nonzero minor (determinant of a maximal submatrix) D of a Newton matrix M. Then D is a nontrivial multiple of the sparse resultant R.*

Proposition 5.3 *Let an $a \times c$ matrix M be a Newton matrix, $a \geq c$, and assume that L is a unit $a \times a$ lower triangular Toeplitz matrix, with its subdiagonal entries randomly chosen from a fixed finite set T, as in lemma 3.1. Then the matrix $W = [O, I_c]LM$ is nonsingular and $\det W$ is a nonzero multiple of the sparse resultant R with a probability at least $1 - c/|T|$.*

Proof Since M has full rank, it follows, by lemma 3.1, that $\det W \neq 0$ with probability at least $1 - c/|T|$. Let $L = [l_{ij}]$ for $i, j = 1, \ldots, a$ (where $l_{ii} = 1$ and $l_{ij} = 0$ if $i < j$). Suppose that the ith row of M expresses polynomial $x^{b(i)} f_{t(i)}$, where $b(i)$ lies in $B_{t(i)}$ and $t(i) \in \{1, \ldots, n+1\}$, for $i = 1, \ldots, a$. Then the kth row of LM for $k = a-c+1, \ldots, a$ expresses the polynomial

$$x^{b(k)} f_{t(k)} + \sum_{j=1}^{k-1} l_{kj} x^{b(j)} f_{t(j)},$$

where $t(j) \in \{1, \ldots, n+1\}$, $b(j) \in B_{t(j)}$. Now the proof of theorem 5.2 applies to W. □

Another important property of Newton matrix M is that its *premultiplication* by a vector reduces to computing the sum of *polynomial products*. Refer to expression (2); both vectors $[g_1, \cdots, g_{n+1}]$ and $[\sum_i g_i f_i]$ are understood with respect to the fixed monomial basis defined by the sequence of monomials indexing the columns. Moreover, any vector can be decomposed into subvectors of length $|B_i|$, $i = 1, \ldots, n+1$, and the respective entries can be thought of as the coefficients of a polynomial g_i. Then the premultiplication by any vector is reduced to computing $\sum_i g_i f_i$.

Example 5.4 *Let $f_1 = c_0 + c_1 x_1 + c_2 x_1 x_2$, with ordered support $A_1 = ((0,0),(1,0),(1,1))$, let $B_1 = ((0,0),(1,0))$ and consider the subsequence $((0,0),(1,0),\ (1,1),\ (2,0),(2,1))$ of S. For an arbitrary row vector $[s_0, s_1, \ldots]$, the first entries can be thought of as the coefficients of polynomial $g_1 = s_0 + s_1 x_1$. Then, premultiplication by this vector starts as follows:*

$$\begin{array}{c} \begin{array}{cccc} 1 & x_1 & & \end{array} \\ \left[\begin{array}{cccc} s_0 & s_1 & \cdots & \end{array}\right] \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & x_1 & x_1x_2 & x_1^2 & x_1^2x_2 \end{array} \\ \left[\begin{array}{ccccc} c_0 & c_1 & c_2 & 0 & 0 \\ 0 & c_0 & 0 & c_1 & c_2 \\ & & \vdots & & \end{array}\right] \end{array} \begin{array}{c} f_1 \\ x_1 f_1 \\ \vdots \end{array} =$$

$$= \begin{array}{c} \begin{array}{ccccc} 1 & x_1 & x_1x_2 & x_1^2 & x_1^2x_2 \end{array} \\ \left[\begin{array}{ccccc} s_0c_0 & s_0c_1 + s_1c_0 & s_0c_2 & s_1c_1 & s_1c_2 \end{array}\right] \end{array} + \cdots.$$

To the right of the matrix, we mark the polynomials filling in the rows, and above the matrix and the vectors, we show the monomials indexing the columns or entries, respectively.

Proposition 5.5 *Let M be an $a \times c$ Newton matrix of the transformation of (2) where $a \geq c$, and let v be a $1 \times a$ vector, both with constant entries. Then computing the vector vM takes $O^*(cn + n\sqrt{d})$ ops and $O(cn)$ storage space, where d is the maximum degree of $f_1, \cdots, f_{n+1}$ in any one variable.*

Proof Consider v as expressing the coefficients of the polynomials g_i whose supports B_i define the rows of M, $i = 1, \ldots, n+1$. Then vM is the row vector expressing $\sum_i f_i g_i$, whose support is $S = \cup_i (A_i + B_i)$, where $A_i = \mathrm{supp}(f_i)$. Moreover, S is precisely the set of the monomials indexing the columns of M, hence $c = |S|$. By theorem 4.7, we can compute $\sum_i f_i g_i$ in $O^*(cn + n\sqrt{d})$ ops.

Since each product requires $O(cn)$ space, and the sum can be stored and updated in $O(c)$ memory locations, the overall storage needed is $O(cn)$. □

An interesting extension for polynomial system solving is when the matrix entries are univariate polynomials in a fixed indeterminate [17, 5]. This means that in the course of performing the computations above, a typical vector by which M is premultiplied has entries that are polynomials in this indeterminate. This would increase the time complexity by an additional quasi-linear factor in the maximum degree of the input polynomials in this indeterminate.

6 Incremental matrix construction

In this section we sketch the incremental algorithm for constructing a Newton matrix proposed in [7] and reduce its time complexity to quasi-quadratic in the matrix dimension, whereas the original algorithm had cubic complexity. The incremental construction yields the smallest Newton matrices among all existing algorithms and, moreover, constructs optimal matrices in several cases, including all cases where optimal matrices provably exist. An implementation is available at the `http` address of the first author. Experiments have shown that the matrix dimension is typically within a factor of three of the optimal.

The matrix is constructed by adding integer points to the candidate sets B_i, until a Newton matrix is found. For every intermediate candidate matrix with at least as many rows as columns, the algorithm tests whether it has full rank. To formalize, let Q_i denote the Newton polytope of f_i and define Minkowski sums $Q_{-i} = \sum_{j\neq i} Q_j$, $i = 1, \ldots, n+1$, and $Q = \sum_j Q_j$. Then the set of row monomials is the disjoint union of sets $B_i \subset Q_{-i} \cap \mathbf{Z}^n$. The set of column monomials always lies in Q and, at any stage of the algorithm, it is defined to be $\cup_i (A_i + B_i)$ for the B_i at this stage. The algorithm linearly orders all points in each Q_{-i}, so that there is a well-defined rule for incrementing the sets B_i for $i = 1, \ldots, n+1$. As the B_i are incremented, the algorithm constructs successively larger matrices until a Newton matrix is found.

Initially B_i contains the optimal number of points, namely $\deg_{f_i} R$, given by identity (1), $i = 1, \ldots, n+1$. The number of incremental steps is bounded by the final number of rows, because every step adds at least one point to some B_i. In practice, every step adds more than one point; in this regard, computing the matrix rank provides useful information and can be done within the same complexity, by corollary 3.5. Most values for the number of tests are smaller than $c/10$.

The matrix obtained at each step is characterized by the same structure as the Newton matrix. The idea is, therefore, to exploit the structure of the rectangular matrix in order to accelerate each rank test.

Lemma 6.1 *Let polynomial system $f_1, \ldots, f_{n+1} \in P$ and let M be an $a \times c$ matrix constructed in the course of the incremental algorithm, with numeric entries, such that $a \geq c$. Testing whether M has full rank by an exact-arithmetic randomized computation requires $O^*(c^2 n + cn\sqrt{d} + ac)$ ops, where d is the maximum degree of the f_i in any variable, and $O(cn+a)$ storage. The same test performed numerically within some given tolerance value requires $O^*(acn + an\sqrt{d} + a^2)$ ops and has space complexity $O(cn + a)$.*

Proof The proof follows from theorems 3.3 and 3.4 if we apply proposition 5.5 to bound the cost of a vector-by-matrix multiplication. □

The following theorem gives an *output-sensitive* upper bound on the worst-case complexity of the incremental algorithm. In the rest of this section, we ignore the cost of computing the monomial set indexing the columns of the Newton matrix. For the sake of simplicity, we make the *hypothesis* that $a = O^*(cn)$, which is validated experimentally.

Theorem 6.2 *Assume that the given $n+1$ polynomials in n variables have numeric coefficients and let t be the number of rank tests required by the incremental algorithm of [7] in order to construct M. Assume that the maximum degree in any variable is $d = O(c^2)$. Then, using the randomized algorithm of theorem 3.3, the time complexity is $O^*(c^2 nt)$, and using the numerical algorithm of theorem 3.4 with some given tolerance, it is $O^*(acnt)$. The space complexity is $O^*(cn)$ for both approaches.*

The previous time complexity bound was $O(a^2 c)$ from [7, lem. 7.2] and the space complexity was $O(ac)$. These bounds follow from the fact that the algorithm tests the nonsingularity of several matrix candidates, by applying an incremental version of LU-decomposition, which is performed in place. Table 2 shows the various parameters in examples studied in [7, 6]. The first three are multihomogeneous systems with 3 groups of 2, 1 and 1 variables respectively, where the corresponding degrees are given in the table, the following two are different expressions of the cyclic 6-root problem, and the last example is the Stewart platform from robot kinematics.

Table 2: Performance of the incremental algorithm

type	n	$\deg R$	c	a	t
$(2,1,1;2,1,1)$	4	240	260	260	5
$(2,1,1;2,2,1)$	4	480	592	690	43
$(2,1,1;2,2,2)$	4	960	1120	1200	≤ 49
original cyclic	6	290	849	1457	180
improved cyclic	5	66	102	121	18
Stewart platform	6	214	405	526	68

Clearly, an important issue concerns a formal bound on the number of singularity tests t.

Lemma 6.3 *Consider the incremental algorithm described above and suppose that a valid Newton matrix is defined by point sets B_i, $i = 1, \ldots, n+1$. If any or all of the B_i are incremented (by following the ordering on the respective set $Q_{-i} \cap \mathbf{Z}^n$), then the new matrix is again a valid Newton matrix.*

This lemma suggests the following heuristic rule to minimize t: At every incremental step, the algorithm adds at least $\deg R$ new rows, by adding as many points in the corresponding sets B_i. Let a_1 denote the number of rows in the first full-rank matrix encountered by the algorithm, and let $a_0 < a_1$ be the number of rows in the last (hence largest) rejected candidate matrix. The algorithm tries to optimize the number of rows by performing a *binary search in the heuristic range* $(a_0, a_1]$. Hence, the total number of tests is roughly $a_1/\deg R + \log \deg R$. We applied the new algorithm to the 1st, 4th and 5th inputs in table 2 and obtained a matrix with the same number of columns after 7, 15 and 5 tests respectively. Based on experimental evidence we can bound a_1 in terms of $\deg R$.

Corollary 6.4 *In the context of theorem 6.2, assume that the number of rows a in a Newton matrix constructed by the*

incremental algorithm is bounded by a constant multiple of $\deg R$. *Then, with the binary search in the heuristic range just described, the time complexity of the algorithm for finding this matrix becomes* $O^*(c^2n)$ *or* $O^*(acn)$, *depending on the rank-test algorithm used.*

There are two main reasons for constructing Newton matrices. The first is for solving systems of nonlinear polynomial equations. We have examined the phase of matrix construction, which is comparatively costly. Once this is over, certain matrix operations are applied to simplify the linear algebra problem [6]; here, the matrix structure comes very handy. The final problem is an eigenvalue/eigenvector computation, for which strong results exploiting matrix structure are desirable.

The second major application is in computing the exact sparse resultant polynomial, which divides the determinant of the Newton matrix. In this context, the coefficients are typically polynomials in a single variable, denoted u. This may be the same situation as in the u-resultant approach [17] or when u has been chosen among the input variables to be "hidden" in the coefficient field [5]. In both cases, the first question is to compute $\det M(u)$ as a univariate polynomial; the rest of the problem is considered in the next section.

Corollary 6.5 *We are given* $a \times c$ *Newton matrix with univariate entries* M *of degree* d. *Under the hypotheses of theorem 6.2, we can compute the univariate determinant of* $c \times c$ *matrix* $W = [0, I_c]LM$ *by using* $O(c^3nd)$ *ops and* $O(cn+cd)$ *storage space.*

Proof This can be achieved by the well-known evaluation-interpolation technique. Since the degree of $\det W$ in u is bounded by cd, the number of evaluations is cd. By corollary 3.5, one determinant computation of the specialized matrix requires $O^*(c^2n)$ ops. The needed space is $O(cn)$ in addition to $O(cd)$ needed for storing the determinant values and interpolating from them to the polynomial coefficients. □

For the u-resultant construction, $d = 1$ and the number of columns containing u equals the degree of resultant R in the coefficients of the u-polynomial. If the latter is f_{n+1}, then the time complexity becomes $O^*(c^2n \deg_{f_{n+1}} R)$.

7 Sparse resultant computation

This section focuses on computing the sparse resultant from a set of Newton matrices, when all input coefficients are given specific numeric values. Then, it is straightforward to extend our algorithms to the case where some polynomial coefficients remain indeterminate or are expressed in terms of parameters, just as at the end of the previous section. Exploiting the matrix structure enables us to decrease the overall complexity by about a linear factor in matrix size.

We shall require an additional property for the Newton matrices used. Associate each matrix with one of the given polynomials f_i, so that the number of rows of M containing multiples of f_i is precisely $\deg_{f_i} R$, hence the degree of $\det M$ in the coefficients of f_i equals the corresponding degree of the resultant. This property can be guaranteed in the case of the incremental algorithm if we fix set B_i to its initial size [7], and is also satisfied in the case of the subdivision-based algorithm of [2]. Thus, either algorithm can be used in the discussion that follows.

The naive way to compute R as the Greatest Common Divisor (GCD) of $n+1$ determinants is known not to work for arbitrary coefficient specializations [19]. For this, two probabilistic methods have been proposed in [2, sect. 5].

Let M_i be the Newton matrix associated to f_i, for $1 \leq i \leq n+1$. Recall that the f_i have indeterminate coefficients and let g_i be the specialization of f_i and h_i be a random polynomial with the same support. Denote by $D_i^{(j)}$, $0 \leq j \leq n+1$ the determinant of matrix M_i for the system obtained after specializing $f_k \mapsto g_k + \epsilon h_k$, for $k \leq j$ and $f_k \mapsto h_k$, for $k > j$, where ϵ is an indeterminate that will go to zero.

The *division method* determines the resultant of the system $g_i + \epsilon h_i$ (within a scalar factor) by

$$R(g_i + \epsilon h_i) = \frac{D_{n+1}^{(n+1)}}{D_{n+1}^{(n)}} \cdots \frac{D_1^{(1)}}{D_1^{(0)}}.$$

The desired resultant $R(g_i)$ can then be obtained by setting $\epsilon = 0$, provided that the choice of h_i is sufficiently generic. This happens with very high probability. Note that R may be computed by using less than $n+1$ matrix determinants, if at least one of them happens to have the same degree as R in the coefficients of more than one polynomial.

Theorem 7.1 *Suppose that we have already computed all (and at most* $n+1$*) necessary Newton matrices for polynomial system* $g_1, \ldots, g_{n+1} \in P$, *with matrix size* $c \times c$. *The sparse resultant of the specialized system can be computed by the division method in* $O^*(c^2n^2 \deg R)$ *ops, using* $O(cn)$ *additional storage space, where* $\deg R$ *denotes the total degree of the sparse resultant in the input coefficients.*

Proof The evaluation-interpolation scheme is used with $1+\deg R$ different values for ϵ, since the degree of $R(g_i+\epsilon h_i)$ in ϵ is bounded by $\deg R$. The dominant complexity is that of evaluating the $2n$ determinants. Since $\deg R \leq c$, the storage for the interpolation phase is in $O(cn)$ by setting $d = 1$ in corollary 6.5. □

The previous time complexity bound was $O^*(M(c) \deg R)$, where $M(c)$ is the arithmetic complexity of $c \times c$ matrix multiplication. Observe that only the constant term of $R(g_i + \epsilon h_i)$ is needed.

The following method uses only two Newton matrix determinants by distinguishing an exponent vector $a \in A_1$ and imposing a related technical constrain on B_1 (for details, see [2, sect. 5]). The first determinant, denoted D_1, is $D_1^{(n+1)}$ under the above notation. The second, denoted D_1', is the determinant of M_1 for specialized system $f_1 \mapsto x_1^a + \epsilon h_1$, $f_i \mapsto g_i + \epsilon h_i$, for $i \geq 2$. Then, the *GCD method* computes

$$R(g_i + \epsilon h_i) = \frac{D_1}{\gcd(D_1, D_1')},$$

and the desired resultant is again obtained by setting $\epsilon = 0$.

Theorem 7.2 *With the above notation, the sparse resultant of* $g_1, \ldots, g_{n+1} \in P$ *can be computed by the GCD method in* $O^*(c^3n)$ *ops, using* $O(cn)$ *total space.*

Proof The dominant step is the computation of D_1, D_1' as univariate polynomials in ϵ, with degree bounded by c. By the evaluation-interpolation scheme, this takes $O^*(c^3n)$ ops and $O(cn)$ storage. Computing the GCD, then evaluating the fraction and, lastly, interpolating to the least significant coefficient of $R(g_i+\epsilon h_i)$, all have dominated complexities. □

The previous time complexity bound was $O^*(M(c)c)$. Note that the univariate GCD computation can be reduced to a branch-free computation of subresultants because the degree of the GCD, which is precisely the extraneous factor in D_1, is known in advance. Moreover, this computation can be enhanced by probabilistic interpolation techniques [19, ch. 15].

Both the division and the GCD method are readily extended to computing the sparse resultant polynomial, if the coefficients are specialized in terms of one or more indeterminates. This covers also the case of the u-resultant.

8 Conclusion

Most complexity bounds rely on the efficiency of FFT, i.e., its quasi-linear time complexity and linear space complexity. Yet, it is known that the latter algorithm is truly advantageous only for rather large inputs, due to its high constant factor. Our methods can be adapted to other basic algorithms for polynomial multiplication of intermediate speed, namely the so-called Karatsuba's method [11], which may be preferable for inputs of moderate size. Karatsuba's multiplication algorithm has linear space complexity and time complexity $O(k^{\lg 3})$ for k-degree polynomials, where lg denotes the logarithm in base 2. See table 1 for some ramifications.

Resultant matrices reduce polynomial system solving in the zero-dimensional case to a linear algebra problem, including an eigenvalue/eigenvector computation. Here strong results exploiting matrix structure are desirable. This is an important open question, so far settled only for symmetric Toeplitz matrices. We may try to combine other ways of exploiting structure and, in particular, the large number of zero entries which usually constitute the great majority. Last but not least, we would like to use information between successive rank tests since every rejected candidate is a submatrix of the next.

Acknowledgments

The first author was partially supported by European ESPRIT project FRISCO (LTR 21.024) and acknowledges enlightening car commutes with Bernard Mourrain. The second author was supported by NSF Grants CCR 9020690 and CCR 9625344, and PSC-CUNY Awards Nos. 666327 and 667340. Work partially conducted while the second author was on sabbatical at INRIA Sophia-Antipolis.

References

[1] Bini, D., and Pan, V. *Polynomial and Matrix Computations*, vol. 1: Fundamental Algorithms. Birkhäuser, Boston, 1994.

[2] Canny, J., and Emiris, I. An efficient algorithm for the sparse mixed resultant. In *Proc. Intern. Symp. on Applied Algebra, Algebraic Algor. and Error-Corr. Codes, Lect. Notes in Comp. Science 263* (Puerto Rico, 1993), G. Cohen, T. Mora, and O. Moreno, Eds., Springer, pp. 89–104.

[3] Canny, J., Kaltofen, E., and Lakshman, Y. Solving systems of non-linear polynomial equations faster. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation* (1989), pp. 121–128.

[4] Canny, J., and Pedersen, P. An algorithm for the Newton resultant. Tech. Rep. 1394, Comp. Science Dept., Cornell University, 1993.

[5] Emiris, I. On the complexity of sparse elimination. *J. Complexity 12* (1996), 134–166.

[6] Emiris, I. A general solver based on sparse resultants: Numerical issues and kinematic applications. Tech. Rep. 3110, INRIA Sophia-Antipolis, France, Jan. 1997.

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

[8] Emiris, I., and Pan, V. Techniques for exploiting structure in matrix formulae of the sparse resultant. *Calcolo, Spec. Issue on Workshop on Toeplitz Matrices, Cortona* (1997). To appear.

[9] Kaltofen, E., and Lakshman, Y. Improved sparse multivariate polynomial interpolation algorithms. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation* (1988), vol. 358 of *Lect. Notes in Comp. Science*, Springer, pp. 467–474.

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

[11] Karatsuba, A., and Ofman, Y. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl. 7* (1963), 595–596.

[12] Manocha, D. Solving systems of polynomial equations. *IEEE Comp. Graphics and Appl., Special Issue on Solid Modeling* (1994), 46–55.

[13] Mourrain, B., and Pan, V. Solving special polynomial systems by using structured matrices and algebraic residues. In *Proc. Workshop on Foundations of Computational Mathematics* (1997), F. Cucker and M. Shub, Eds., Springer, pp. 287–304.

[14] Pan, V. Numerical computation of a polynomial GCD and extensions. Tech. Rep. 2969, INRIA, Sophia-Antipolis, France, Aug. 1996.

[15] Pan, V. Parallel computation of polynomial GCD and some related parallel computations over abstract fields. *Theor. Comp. Science 162*, 2 (1996), 173–223.

[16] Sturmfels, B. On the Newton polytope of the resultant. *J. of Algebr. Combinatorics 3* (1994), 207–236.

[17] van der Waerden, B. *Modern Algebra*, 3rd ed. F. Ungar Publishing Co., New York, 1950.

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

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