Efficient Algorithms for MultiPolynomial Resultant

Dinesh Manocha*

Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA

The multipolynomial resultant of a set of equations is fundamental in quantifier elimination over the elementary theory of real and algebraically closed fields. Earlier algorithms for resultant computation and symbolic elimination are considered slow in practice. In this paper we present efficient algorithms to compute multipolynomial resultants and demonstrates their use for polynomial manipulation and symbolic applications. The algorithms utilize the linear algebra formulation of the resultants and combines it multivariate interpolation and modular arithmetic for fast computation. It is currently been implemented as part of a package and we discuss its performance as well.

Received February 1993, revised May 1993

1. INTRODUCTION

The problem of eliminating a set of variables from a system of polynomial equations is fundamental in computer algebra. Given a system of polynomial equations, their multipolynomial resultant is a function of the coefficients of the equations obtained after eliminating the variables [33, 43, 48]. The computation of the multipolynomial resultant of a system of polynomial equations has an important application, among many others, in the quantifier elimination over the elementary theory of real and algebraically closed fields [8, 41, 23].

There are two other techniques known in the literture for eliminating a set of variables. They are Gröbner bases and Ritt-Wu's algorithm. The algorithm for Gröbner bases generates special bases for polynomial ideals and was originally formulated by Buchberger [5, 6]. Eliminating a set of variables is a special application of Gröbner bases. Ritt-Wu's algorithm for variable elimination has been developed by Wu Wen-Tsün [51] using an idea proposed by Ritt [42]. This approach is based on Ritt's characteristic set construction and successfully applied to automated geometry theorem proving by Wu. All these methods of elimination are surveyed in [28].

A great deal of work has been done in the literature on the theory, implementation and application of these methods for elimination. However, in the last decade most of the effort has been directed towards use of Gröbner bases. Most computer algebra systems, like Mathematica, Maple, Reduce etc. have a package for computing the Gröbner bases of an ideal. Motivated by the need of faster implementations, some special systems have been developed exclusively for Gröbner bases computation, like Macaulay and Cocoa. As far as resultants are concerned, many different formulations are known for sets of polynomials. In practice, almost all computer algebra systems have a routine for the ubiquitous Sylvester resultant of two polynomial equations. However, all these implementations have been found to be very slow for geometric applications on current hardware. Trying to compute the Gröbner bases of $3-4$ polynomials in $4-5$ variables of moderate degrees can take an incredible amount of time, if the system is able to compute. Many a times the computation goes on until the machine runs out of all the virtual space (which is of the order of few gigabytes). This fact was highlighted in the context of implicitizing parametric surfaces in geometric modeling applications by Hoffmann [21, 20]. In general, it is regarded that no good algorithms and implementations are known for eliminating variables from a system of polynomial equations.

In this paper we present efficient algorithms to compute multipolynomial resultants based on techniques from linear algebra. Almost all formulations of resultants express it in terms of matrices and determinants. To compute symbolic determinants we use multivariate interpolation and therefore, reduce the problem to integer number crunching as opposed to manipulating symbolic expression. To speed up the computation we use modular arithmetic. The algorithm is further improved by using a variety of techniques from linear algebra. We also describe its implementation and performance on the implicitization benchmark. The resulting algorithm is fast and it is possible to come up with tight bounds on its running time. This paper is a refined and completed version of some of the results presented in [36, 39].

*This research was supported in part by David and Lucile Packard Fellowship, IBM Graduate Fellowship and a Junior Faculty Award.

The rest of the paper is organized in the following manner: In Section 2, we briefly review the various resultant formulations for sparse and dense polynomial systems. Section 3 analyzes the problem of efficiently computing resultants and reduces the problem to multivariate interpolation. We also review some literature on multivariate interpolation. In Section 4, we improve the algorithms by combining the interpolation algorithm with matrix computations. Section 5 discusses the implementation and we highlight its performance on the problem of implicitizing rational parametric surfaces and compare it with other implementations in Section 6.

2. QUANTIFIER ELIMINATION

Ever since Tarski's work [46] on the decidability of the theory of real closed fields, there have been improvements in the time and space bounds of algebraic algorithms. Collins presented a double-exponential time algorithm for the theory using cylindrical algebraic decomposition. Collins algorithm was improved by [3] to single-exponential space. The algorithm described in [3] is purely symbolic and does not require numeric root isolation as in Collins algorithm. The main lemma in [3] for the roots of a single univariate polynomial is generalized by Canny using multipolynomial resultants to common zeros of a system of multivariate polynomials [8] and thereby resulting in a PSPACE algorithm for the existential theory of the reals. Renegar has also presented PSPACE decision procedures for the existential theory of real-closed fields [41]. Thus, the best theoretical algorithms for quantifier elimination and the theory of reals are based on multipolynomial resultants.

2.1. MultiPolynomial Resultants

Given n homogeneous polynomials in n unknowns, the resultant of the given system of equations is a polynomial in the coefficients of the given equations. The most familiar form of the resultant is the Sylvester's formulation for the case $n = 2$. In this case, the resultant can always be expressed as determinant of a matrix. However, a single determinant formulation may not exist for any arbitrary n and the most general formulation of resultant (to the best of our knowledge) expresses it as a ratio of two determinants [31]. If all the polynomials have the same degrees, an improved formulation is given in [32]. Many a time both determinants evaluate to zero. To compute the resultant we need to perturb the equations and use limiting arguments. This corresponds to computing the characteristic polynomials of both the determinants [9]. The ratio of the two characteristic polynomials is termed the *Generalized Characteristic Polynomial*, and the resultant corresponds to its constant term [10]. If the constant term is zero, its lowest degree term contains important information and can be used for computing the proper components in the presence of excess components. This formulation has advantages for both numeric and symbolic applications. Many special cases, corresponding to $n = 2, 3, 4, 5, 6$ when the resultant can be expressed as the determinant of a matrix, are given in [15, 24, 40, 35]. Historical accounts of resultants and elimination theory are presented in [1, 49].

Most of the formulation presented in the classical literature correspond to computing the resultants of dense polynomial systems. In the last few decades, a number of results have appeared in the literature pertaining to the resultants of sparse polynomial systems [2, 18, 44]. This is very useful due to the fact that many polynomial systems encountered in applications are rather sparse [34]. Resultant formulations of sparse polynomial systems, expressed in terms of matrices and determinants, have appeared in [11, 45].

3. MULTIVARIATE INTERPOLATION

The algorithms for resultant computation correspond to computing determinant of matrices. The entries of the matrices are the coefficients of the polynomial equations or polynomial functions of the coefficients. In [36], Manocha and Canny use algorithms based on multivariate interpolation to compute the symbolic determinants. In this section, we review various algorithms known in the literature for multivariate interpolation and use for them computing multipolynomial resultants in the following sections.

3.1. Dense Interpolation

Let's assume that each entry of the matrix $\mathbf{M}$ is a polynomial in $x_1, x_2, \ldots, x_n$ and $\mathbf{M}$ is of order $m \times m$. If the entries are rational functions, they can be multiplied by suitable polynomials such that each entry of the resulting matrix is a polynomial. Furthermore, the coefficients of matrix entries are from a field of characteristic zero. The main idea is to determine the power products that occur in the determinant, say a multivariate polynomial $F(x_1, x_2, \ldots, x_n)$. Let the maximum degree of x_i in $F(x_1, x_2, \ldots, x_n)$ be d_i. The d_i's can be determined from the matrix entries. F can have at most $q_1 = (d_1 + 1)(d_2 + 1) \ldots (d_n + 1)$ monomials. In some cases, it is easier to compute a bound on the total degree of F. Any degree d polynomial in n variables can have at most $q_2 = C_R(n, d)$ coefficients, where

$$C_R(n,d) = \begin{pmatrix} n+d-1 \\ d \end{pmatrix}.$$

We represent the determinant as

$$F(x_1, x_2, \ldots, x_n) = c_1 m_1 + c_2 m_2 + \ldots + c_q m_q, \quad (1)$$

where q is bounded by q_1 or q_2. The $m_i = x_1^{d_{1,i}} x_2^{d_{2,i}} \ldots x_n^{d_{n,i}}$ are the distinct monomials and the c_i are the corresponding non-zero coefficients. The problem of determinant expansion corresponds to computing the c_i's. By choosing different substitutions for $(x_1, \ldots, x_n)$ and computing the corresponding $F(x_1, \ldots, x_n)$ (expressed as determinants of numeric matrices) the problem is reduced to that of *multivariate interpolation* [4, 52]. Since there are q monomials, we need to choose q distinct substitutions and solve the resulting $q \times q$ system of linear equations using standard Gaussian elimination. The running time of the resulting algorithm is $O(q^3)$ and takes $O(q^2)$ space. To reduce the running time and space of the algorithm we perform Vandermonde interpolation.

Let $p_1, p_2, \ldots, p_n$ denote distinct primes and $b_i = p_1^{d_{1,i}} p_2^{d_{2,i}} \ldots p_n^{d_{n,i}}$ denote the value of the monomial m_i at $(p_1, p_2, \ldots, p_n)$. Clearly, different monomials evaluate to different values. Let $a_i = F(p_1^i, p_2^i, \ldots, p_n^i)$, $i = 1, q$. a_i's are computed by Gaussian elimination. Thus, the problem of computing c_i's is reduced to solving a transposed Vandermonde system system $\mathbf{Vc} = \mathbf{a}$, where

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ b_1 & b_2 & \ldots & b_q \\ \vdots & \vdots & \vdots & \vdots \\ b_1^{q-1} & b_2^{q-1} & \ldots & b_q^{q-1} \end{pmatrix},$$

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_q \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_q \end{pmatrix}. \quad (2)$$

Computing each a_i takes $O(m^3)$ time, where m is the order of the matrix. As far as solving transposed Vandermonde systems is concerned, simple algorithms of time complexity $O(q^2)$ and $O(q)$ space requirements are known [52]. In Kaltofen and Laksman [27], an improved algorithm of time complexity $O(M(q)log(q))$ is presented, where $M(q)$ is the time complexity of multiplying two univariate polynomials of degree q. Thus, the running time of the resulting algorithm is $O(qm^3 + M(q)log(q))$ and space requirements are $O(m^2 + q)$. However, due to simplicity we use the $O(q^2)$ algorithm for Vandermonde interpolation and the rest of the time complexities are presented in terms of that.

Before we use Vandermonde interpolation, the algorithm for computing symbolic determinant needs to know q and the m_i's corresponding to nonzero c_i's. In the worst case, q may correspond to q_1 or q_2 and the resulting problem is that of *dense interpolation*. The m_i's are enumerated in some order (e.g. lexicographic) and b_i's are computed by substituting p_j for x_j. In general, d is larger than n and q_1 or q_2 tend to grow exponentially with the number of variables and as a polynomial function of the degree of each matrix entry. Furthermore, this approach becomes unattractive when the determinant is a sparse polynomial.

A sparse interpolation algorithm for such problems has been presented by Ben-Or and Tiwari [4]. Its time and space complexity have been improved in [27].

3.2. Sparse Interpolation

Ben-Or and Tiwari's algorithm needs an upper bound $T \geq q$ on the number of non-zero monomials in $F(x_1, \ldots, x_n)$. Given $(p_1^j, p_2^j, \ldots, p_n^j)$, it computes the monomial values $b_i = p_1^{d_{1,i}} p_2^{d_{2,i}} \ldots p_n^{d_{n,i}}$ by computing the roots of an auxiliary polynomial $G(z)$. These b_i's are used for defining the coefficient matrix of the Vandermonde system.

The polynomial $G(z)$ is defined as

$$G(z) = \prod_{i=1}^{T} (z - b_i) = z^k + g_{k-1} z^{k-1} + \ldots + g_0.$$

Its coefficients, g_i's, are computed by solving a Toeplitz system of equations [4]. The Toeplitz system is formed by computing the a_i's and using the property $G(b_i) = 0$. Given $G(z)$, the algorithm computes its integer roots to compute the corresponding to b_i's. The roots are computed using a p-adic root finder [30]. The running time of the resulting algorithm for polynomial interpolation is $(ndT^3 log(n) + m^3 T)$. The dominating step is the polynomial root finder which takes $O(T^3 log(B))$, where B is an upper bound on the values of the roots.

This algorithm is unattractive for expanding symbolic determinants due to the fact that it is rather difficult to come up with a sharp bound on T. We only have the choice of using $T = q_1$ or $T = q_2$ and the resulting problem corresponds exactly to dense interpolation. As a result, Ben-Or and Tiwari's algorithm is not well suited for our application.

3.3. Probabilistic Interpolation

Zippel has presented a probabilistic algorithm for interpolation which does not expect any bound on the non-zero terms of the polynomial being interpolated [52]. It expects a bound on the degree of each variable in any monomial. Such a bound is easy to compute for a symbolic determinant. In fact, this bound is tight for resultant applications. The algorithm proceeds inductively, uses the degree information and develops an estimate of the number of terms as each new variable is introduced. As a result its performance is *output sensitive* and depends on the actual number of terms in the polynomial. We present a brief outline of the algorithm below. It has been explained in detail in [52].

Choose n random numbers $r_1, \ldots, r_n$ from the coefficient field used for defining the polynomial coefficients. The algorithm proceeds inductively and introduces a variable at each stage. Let us assume we are at the kth stage and have interpolated a polynomial in k variables. The resulting polynomial is of the form

$$F_k(x_1, \ldots, x_k) = F(x_1, x_2, \ldots, x_k, r_{k+1}, \ldots, r_n)$$
$$= c_{1,k} m_{1,k} + \ldots + c_{\alpha,k} m_{\alpha,k},$$

where

$$c_{i,k} m_{i,k} = (c_i m_i)_{x_{k+1}=r_{k+1}, x_{k+2}=r_{k+2}, \ldots, x_n=r_n}.$$

In this case, $\alpha \leq q$. F_i is a polynomial in i variables $x_1, \ldots, x_i$.

To compute $F_{k+1}(x_1, \ldots, x_{k+1})$ from $F_k(x_1, \ldots, x_k)$, it represents F_{k+1} as

$$F_{k+1}(x_1, \ldots, x_{k+1}) = F(x_1, \ldots, x_{k+1}, r_{k+2}, \ldots, r_n)$$
$$= h_1(x_{k+1}) m_{1,k} + \ldots + h_\alpha(x_{k+1}) m_{\alpha,k},$$

where each each $h_i(x_{k+1})$ is a polynomial of degree d_{k+1}. d_{k+1} is a bound on the degree of x_{k+1} in any term of $F(x_1, \ldots, x_n)$. It computes $h_i(x_{k+1})$ by Vandermonde interpolation. The value of each $h_i(x_{k+1})$ are obtained for $d_{k+1}+1$ values $1, p_{k+1}, p_{k+1}^2, \ldots, p_{k+1}^{d_{k+1}}$ as follows:

$$F_{k+1}(1, \ldots, 1, p_{k+1}^j, r_{k+2}, \ldots, r_n) =$$
$$h_1(p_{k+1}^j) + \ldots + h_\alpha(p_{k+1}^j).$$
$$F_{k+1}(p_1, \ldots, p_k, p_{k+1}^j, r_{k+2}, \ldots, r_n) =$$
$$h_1(p_{k+1}^j) b_{i,k} + \ldots + h_\alpha(p_{k+1}^j) b_{\alpha,k}$$
$$F_{k+1}(p_1^2, \ldots, p_k^2, p_{k+1}^j, r_{k+2}, \ldots, r_n) =$$
$$h_1(p_{k+1}^j) b_{i,k}^2 + \ldots + h_\alpha(p_{k+1}^j) b_{\alpha,k}^2$$
$$\vdots$$
$$F_{k+1}(p_1^{\alpha-1}, \ldots, p_k^{\alpha-1}, p_{k+1}^j, r_{k+2}, \ldots, r_n) =$$
$$h_1(p_{k+1}^j) b_{i,k}^{\alpha-1} + \ldots + h_{\alpha-1}(p_{k+1}^j) b_{\alpha,k}^{\alpha-1}$$

This is a Vandermonde system of α equations in α unknowns and can be solved for $h_i(p_{k+1}^j)$. The computation is repeated for $j = 0, \ldots, d_{k+1}$. Given

$$h_i(1), h_i(p_{k+1}), h_i(p_{k+1}^2), \ldots, h_i(p_{k+1}^{d_{k+1}}),$$

use Vandermonde interpolation (as that for univariate polynomials) to compute $h_i(x_{k+1})$. These $h_i(x_{k+1})$ are substituted to represent F_{k+1} as a polynomial in $k+1$ variables. $F_{k+1}(x_1, \ldots, x_{k+1})$ can have at most $(\alpha * (d_{k+1}+1)) \leq q$ terms.

The algorithm starts with $F(r_1, \ldots, r_n)$ and computes the n stages as shown above. There is a small chance that the answer produced by the algorithm is incorrect. This happens for a choice of $r_1 \ldots, r_n$ and can be explained in the following manner. Let the stage I of the algorithm result in a polynomial of the form

$$F_1(x_1) = F(x_1, r_2, \ldots, r_n)$$
$$= c_0 + c_1 x_1^3 + \ldots + c_\beta x_1^{d_1}.$$

$F_2(x_1, x_2)$ actually is a polynomial of the form

$$F_2(x_1, x_2) = F(x_1, x_2, r_3, \ldots, r_n)$$
$$= h_1(x_2) + h_2(x_2) x_1 + h_3(x_2) x_1^2 + \ldots + h_{d_1+1}(x_2) x_1^{d_1}.$$

This implies

$$h_1(r_2) = c_0, \quad h_2(r_2) = 0,$$
$$h_3(r_2) = 0, \ldots, h_{d_1+1}(r_2) = c_\beta.$$

In practice, $h_2(x_2)$ and $h_3(x_2)$ maybe nonzero polynomials, but for the choice of r_2, where $h_2(r_2) = 0$, $h_3(r_2) = 0$, the algorithm assumes them as zero polynomials and proceeds. As a result the algorithm may report fewer terms in $F(x_1, \ldots, x_n)$.

Such an error is possible at each stage of the algorithm. Let us assume that r_i's are chosen uniformly randomly from a set of size S, than the probability that this algorithm gives a wrong answer is less than

$$\frac{nd^2q^2}{S}, \tag{3}$$

where $d = \max(d_1, \ldots, d_n)$ [52].

Later on we use this probability bound for the choice of finite fields used for modular computation. The running time of the algorithm is $O(ndq^2 + m^3 ndq)$.

A deterministic version of this algorithm of complexity $O(ndq^2 T + m^3 ndqT)$ is given in [52] as well. T is an upper bound on the number of non-zero terms in the polynomial. In the worst case, T corresponds to q_1 or q_2. However, the resulting algorithm has better average complexity than the deterministic sparse or dense interpolation algorithms. Due to simplicity and low time complexity of the algorithm, we use the probabilistic version for our implementation. However, we give the user the flexibility of imposing a bound on the probability of failure and choose the finite fields appropriately. Furthermore, we introduce simple, randomized checks in the algorithm to detect wrong answers. These checks correspond to substituting random values for $x_1, \ldots, x_n$ in the matrix entries and computing the resulting numeric determinant. We verify the result by substituting the same values for $x_1, \ldots, x_n$ in the polynomial obtained after interpolation and comparing it with numeric determinant.

4. IMPROVED INTERPOLATION ALGORITHMS

In this section, we present improved algorithms for computing symbolic determinants using multivariate polynomial interpolation. We use a variety of techniques from *linear algebra* to decrease the number of function evaluations and the symbolic complexity of the resulting computation. These include linearizing a matrix polynomial to an equivalent block companion matrix followed by similarity transformations for reduction to upper Hessenberg form and finally to Frobenius canonical form. These transformations are used along with probabilistic and dense interpolation algorithms and as a result the interpolation problem is divided into smaller problems of lower complexity. They also lead to improved and faster algorithms for solving system of non–linear equations based on u–resultants as highlighted in [9] and computing the generalized characteristic polynomials for a system of equations [10].

4.1. Linear Algebra

Algorithms for resultant computation deal with matrices and determinants. As a result, we utilize many properties from linear algebra for efficient resultant computation. In this section we review some techniques from linear algebra and use them in the following sections. More details are given in [19, 50].

DEFINITION 4.1. (MATRIX POLYNOMIALS) *If $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_k$ are $m \times m$ numeric matrices on rationals, then the matrix-valued function defined on the rationals by $\mathbf{L}(\lambda) = \Sigma_{i=0}^{k} \mathbf{A}_i \lambda^i$ is called a* matrix polynomial *of degree k. When $\mathbf{A}_k = \mathbf{I}$, the identity matrix, the matrix polynomial is said to be* monic.

Let us consider the case when $\mathbf{A}_k$ is a non–singular matrix. Let

$$\overline{L}(\lambda) = \mathbf{A}_k^{-1}\mathbf{L}(\lambda), \text{ and } \overline{\mathbf{A}}_i = \mathbf{A}_k^{-1}\mathbf{A}_i, \text{ for } 0 \le i < k.$$

$\overline{L}(\lambda)$ is a monic matrix polynomial. According to Theorem 1.1 [19], the determinant of $\overline{L}(\lambda)$ corresponds exactly to the characteristic polynomial of

$$\mathbf{C} = \begin{bmatrix} \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_m & \ldots & \mathbf{0} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I}_m \\ -\overline{\mathbf{A}}_0 & -\overline{\mathbf{A}}_1 & -\overline{\mathbf{A}}_2 & \ldots & -\overline{\mathbf{A}}_{k-1} \end{bmatrix}, \quad (4)$$

where $\mathbf{0}$ and $\mathbf{I}_m$ are $m \times m$ null and identity matrices, respectively. We refer $\mathbf{C}$ as the *block companion matrix* associated with the matrix polynomial $\overline{\mathbf{L}}(\lambda)$. In other words, the eigenvalues of $\mathbf{C}$ correspond exactly to the roots of Determinant($\mathbf{L}(\lambda)$) = 0 or Determinant($\overline{\mathbf{L}}(\lambda)$) = 0

In the case where $\mathbf{A}_k$ is a singular matrix, the matrix polynomial $\mathbf{L}(\lambda)$ is linearized into a *companion polynomial* $\mathbf{C}_L(\lambda)$ of the form [19]:

$$\mathbf{C}_L(\lambda) = \begin{bmatrix} \mathbf{I}_m & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{A}_k \end{bmatrix} \lambda + \begin{bmatrix} \mathbf{0} & -\mathbf{I}_m & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I}_m & \ldots & \mathbf{0} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & -\mathbf{I}_m \\ \mathbf{A}_0 & \mathbf{A}_1 & \ldots & \mathbf{A}_{k-2} & \mathbf{A}_{k-1} \end{bmatrix}. \quad (5)$$

We will represent this linearization as $\mathbf{P}\lambda + \mathbf{Q}$. The determinant of $\mathbf{C}_L(\lambda) = \mathbf{P}\lambda + \mathbf{Q}$ corresponds exactly to the determinant of $\mathbf{L}(\lambda)$.

The roots of the characteristic polynomial of a matrix correspond to its eigenvalues. Moreover, the eigenvalues of a matrix $\mathbf{M}$ are invariant under similarity transformations of the form $\overline{\mathbf{M}} = \mathbf{H}^{-1}\mathbf{M}\mathbf{H}$, where $\mathbf{H}$ is a non-singular matrix. As a result the characteristic polynomial of the matrix obtained after applying similarity transformations to $\mathbf{C}$ in (4) corresponds exactly to the determinant of $\overline{\mathbf{L}}(\lambda)$.

DEFINITION 4.2. (UPPER HESSENBERG MATRIX) *An upper Hessenberg matrix is of the form*

$$\mathbf{S} = \begin{bmatrix} s_{1,1} & s_{1,2} & \ldots & s_{1,m-1} & s_{1,m} \\ s_{2,1} & s_{2,2} & \ldots & s_{2,m-1} & s_{2,m} \\ 0 & s_{3,2} & \ldots & s_{3,m-1} & s_{3,m} \\ 0 & 0 & \ldots & \ldots & s_{4,m} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & s_{m,m-1} & s_{m,m} \end{bmatrix}$$

An upper Hessenberg matrix is reduced, *if any of the subdiagonal elements, $s_{i,i-1}$, is zero.*

Frobenius Canonical Form Let the characteristic polynomial of $\mathbf{M}$ be

$$(-1)^m[\lambda^m - c_{m-1}\lambda^{m-1} - c_{m-2}\lambda^{m-2} - \ldots - c_0] = 0.$$

Then the characteristic polynomial of the matrix

$$\mathbf{C} = \begin{bmatrix} c_{m-1} & c_{m-2} & \ldots & c_1 & c_0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix}$$

is identical to the characteristic polynomial of $\mathbf{M}$. The matrix $\mathbf{C}$ is the *companion matrix* of the characteristic polynomial of $\mathbf{M}$. A matrix may sometimes be similar to the companion matrix of its characteristic polynomial. The exact criterion for similarity is defined in terms of the multiplicity of the eigenvalues of $\mathbf{M}$ and the structure of its Jordan canonical form [50]. A matrix is called *derogatory* if it is not similar to the companion matrix of its characteristic polynomial.

A *Frobenius matrix* $\mathbf{C}_r$ of order r is a matrix of the form

$$\mathbf{C}_r = \begin{bmatrix} c_{r-1} & c_{r-2} & \dots & c_1 & c_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \qquad (6)$$

Every matrix $\mathbf{M}$ may be reduced by a similarity transformation to the direct sum of a number s of Frobenius matrices denoted as $\mathbf{C}_{r_1}, \mathbf{C}_{r_2}, \dots, \mathbf{C}_{r_s}$. The characteristic polynomial of each $\mathbf{C}_{r_i}$ divides the characteristic polynomial of all the preceding $\mathbf{C}_{r_j}$. In the case of a non–derogatory matrix, $s = 1$ and $r_1 = n$. The direct sum of the Frobenius matrices is called the *Frobenius canonical form*. The transformed matrix is represented as

$$\mathbf{F} = \begin{bmatrix} \mathbf{C}_{r_1} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_{r_2} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{C}_{r_{s-1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{C}_{r_s} \end{bmatrix},$$

where $\mathbf{0}$ denotes a null matrix of appropriate order.

4.2. Univariate Determinants

In this section we consider the problem of expanding a univariate symbolic determinant. Each entry of the $m \times m$ matrix, $\mathbf{L}(\lambda)$, is a polynomial of degree k in λ and let the determinant be a polynomial of degree d. The degree d is bounded by $n = km$. A simple algorithm is evaluating the numeric determinant for $\lambda = p^i$ for $0 \leq i \leq d$, where p is a prime number. This is followed by Vandermonde interpolation to compute the symbolic determinant. However, the running cost of the method is $O(dm^3 + d^2)$ finite field operations, although the cost of interpolation can be improved by using Kaltofen and Laksman's algorithm [27]. In many applications of resultants, each entry of the matrix is a linear polynomial in the coefficients of the nonlinear equations and therefore, $d \leq m$. As a result, the running time of the algorithm is $O(d^4)$ finite field operations. In this section we use techniques from linear algebra and highlight a simple and improved algorithm of $O(d^3 + n^3)$ complexity for most cases in practice. This algorithm is repeatedly used in the multivariate interpolation algorithms. We refer to the algorithm as *univariate determinant algorithm*. The algorithm is:

1. Express $\mathbf{L}(\lambda) = \Sigma_{i=0}^{k} \mathbf{A}_i \lambda^i$ as a matrix polynomial.
2. Linearize the matrix polynomial to an equivalent companion matrix $\mathbf{C}$ of order $d \times d$ using the reduction presented in (4).
3. Use similarity transformations to construct an upper Hessenberg matrix, $\mathbf{S}$, similar to $\mathbf{C}$.
4. Transform $\mathbf{S}$ to its Frobenius canonical form using similarity transformations.
5. The characteristic polynomial of $\mathbf{C}$, $P(\lambda)$, is computed by the product of the of the characteristic polynomials of Frobenius matrices constituting the Frobenius canonical form.
6. Multiply $P(\lambda)$ by the determinant of the leading matrix of $\mathbf{L}(\lambda)$.

The linearization of a matrix polynomial into a companion follows from (4), in case the leading matrix $\mathbf{A}_k$ is non–singular. Lets consider the case when the leading matrix is singular.

We use the reduction, (5), to linearize the matrix polynomial into a companion matrix of the form $\mathbf{P}\lambda + \mathbf{Q}$. This matrix can be transformed into an equivalent companion matrix using row operations, similar to the ones used in Gaussian elimi nation.

Let the rows of $\mathbf{P}$ and $\mathbf{Q}$ be represented as $\mathbf{p}_i$ and $\mathbf{q}_j$, respectively. Each matrix is of order n. Moreover, their elements are denoted as p_{ij} and q_{ij}. We perform elementary row operations on $\mathbf{P}$ to reduce it to an upper triangul ar form. Each elementary row operation is of the form:

$$\mathbf{p}_k = \mathbf{p}_k - \frac{p_{ki}}{p_{ii}} \mathbf{p}_i.$$

At each step perform a corresponding operation is performed on $\mathbf{Q}$.

$$\mathbf{q}_k = \mathbf{q}_k - \frac{p_{ki}}{p_{ii}} \mathbf{q}_i,$$

As a result the determinant of $\mathbf{P}\lambda + \mathbf{Q}$ remains a constant, where $\mathbf{P}$ and $\mathbf{Q}$ are updated with elementary row operations highlighted above (as in Gaussian elimination).

After the row operations $\mathbf{P}$ and $\mathbf{Q}$ are of the form

$$\begin{aligned} \mathbf{P} &= \begin{bmatrix} \mathbf{P}_1 & \mathbf{P}_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \qquad (7) \\ \mathbf{Q} &= \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{Q}_3 & \mathbf{Q}_4 \end{bmatrix}, \end{aligned}$$

where $\mathbf{P}_1$ and $\mathbf{Q}_1$ are square matrices of order n_1, $\mathbf{P}_2$ and $\mathbf{Q}_2$ are matrices of order $n_1 \times n - n_1$. $\mathbf{P}_1$ is an upper triangular matrix. Moreover, $\mathbf{Q}_4$ and the corresponding null matrix in $\mathbf{P}$ are square matrices of order $n - n_1$.

Let us consider the case when $\mathbf{Q}_4$ is a singular matrix. Lets construct the matrix $\mathbf{R} = [\mathbf{Q}_3\ \mathbf{Q}_4]$ of order $n_1 \times n$. Using elementary row operations compute the rank of $\mathbf{R}$. Let the rank be r and represent its submatrix structure as

$$\mathbf{R} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{Q}_3' & \mathbf{Q}_4' \end{bmatrix},$$

where $\mathbf{Q}_4'$ is a non-singular square matrix of order r. In case $n_1 > 0$, $r = 0$, $\mathbf{P}\lambda + \mathbf{Q}$ is a singular pencil. The rest of the algorithm involves reorganizing the submatrices $\mathbf{P}$ and $\mathbf{Q}$, such that $\mathbf{P}_1$ and $\mathbf{Q}_1$ are square matrices of order $n - r$. The resulting system is similar to $\mathbf{P}$ and $\mathbf{Q}$ in (7), such that $\mathbf{Q}_4$ (which has order r after these row operations and readjustment of elements in the submatrices) is non-singular.

Given $\mathbf{P}$ and $\mathbf{Q}$ corresponding to (7), the determinant of $\mathbf{P}\lambda + \mathbf{Q}$ is equal to determinant of

$$\mathbf{G}(\lambda) = \begin{bmatrix} \mathbf{P}_1\lambda + \mathbf{Q}_1 - (\mathbf{P}_2\lambda + \mathbf{Q}_2)\mathbf{Q}_4^{-1}\mathbf{Q}_3 & \mathbf{0} \\ \mathbf{Q}_3 & \mathbf{Q}_4 \end{bmatrix}.$$

Let

$$\mathbf{G}_1\lambda + \mathbf{G}_2 = \mathbf{P}_1\lambda + \mathbf{Q}_1 - (\mathbf{P}_2\lambda + \mathbf{Q}_2)\mathbf{Q}_4^{-1}\mathbf{Q}_3.$$

It follows that

$$\det.(\mathbf{G}(\lambda)) = \det.(\mathbf{G}_1\lambda + \mathbf{G}_2) \times \det.(\mathbf{Q}_4).$$

In case, $\mathbf{G}_1$ is non–singular, the problem can be reduced to an equivalent companion matrix, otherwise the above algorithm for computing a companion matrix of a system of the form $\mathbf{P}\lambda + \mathbf{Q}$ can be used recursively by substituting $\mathbf{G}_1$ and $\mathbf{G}_2$ for $\mathbf{P}$ and $\mathbf{Q}$, respectively. The fact that $\mathbf{G}_1$ and $\mathbf{G}_2$ are matrices of order $n - r$ and $r > 0$ implies that the algorithm terminates after a finite number of steps.

The complexity of the algorithm highlighted can be as high as $O(n^4)$ in the worst case. The method highlighted above is similar to the computation of Kronecker canonical forms in [47]. The algorithm in [47] is targeted towards finite precision arithmetic and deals with numerical issues as well. Since we are performing our computations using finite fields and modular arithmetic, we do not have to worry numerical accuracy issues. Algorithms of complexity $O(n^3)$ to compute the Kronecker canonical forms have been highlighted in [7]. The algorithm in [7] involves rank computation using singular value decompositions and orthogonal transformations. Since we are performing our computations using finite fields and modular arithmetic, we can use elementary row operations for rank computations. As a result, we know that the matrix pencil, $P\lambda + Q$ is either singular, or are able to reduce it to an equivalent companion matrix C, as presented in (4) in $O(n^3)$ steps.

4.3. Dense Interpolation

In this section we extend the univariate symbolic determinant algorithm to interpolate multivariate determinants. Each entry of the $m \times m$ matrix, $\mathbf{M}$, is a polynomial in $x_1, x_2, \ldots, x_n$. The determinant is a polynomial of the form $F(x_1, \ldots, x_n)$ as defined in (1). Furthermore, we know the bounds on the degree of each variable in the resultant. Let the maximum degree of x_i in $F(x_1, x_2, \ldots, x_n)$ be d_i. This degree bound can always be computed by adding the degrees of n_i in the rows or columns of $\mathbf{M}$. Based on these degrees, the bounds on the number of terms, q_1 and q_2, are defined in Section 3.1.

Let's consider the algorithm for dense multivariate interpolation. Without loss of generality, we assume that $d_n = \max(d_1, d_2, \ldots, d_n)$. Express $\mathbf{M}$ as a matrix polynomial in x_m as

$$\mathbf{M} = \mathbf{M}_{\mathbf{d_n}} x_n^{d_n} + \ldots + \mathbf{M}_{\mathbf{1}} x_n + \mathbf{M}_{\mathbf{0}},$$

where $\mathbf{M}_{\mathbf{d_n}}, \ldots, \mathbf{M}_{\mathbf{1}}, \mathbf{M}_{\mathbf{0}}$ are matrices whose elements are polynomial functions in $x_1, \ldots, x_{n-1}$. The rest of the algorithm proceeds iteratively. At the ith iteration:

1. Substitute $(x_1, \ldots, x_{n-1}) = (p_1^i, \ldots, p_{n-1}^i)$ in $\mathbf{M}$ and treat it as a univariate matrix polynomial in x_n.
2. Compute the determinant of the matrix polynomial using the algorithm highlighted in the previous section. The resulting polynomial corresponds to $F(p_1^i, p_2^i, \ldots, p_{n-1}^i, x_n)$.

Let us express F as a polynomial in x_n. That is:

$$F(x_1, x_2, \ldots, x_n) = F_0(x_1, \ldots, x_{n-1}) +$$
$$F_1(x_1, \ldots, x_{n-1})x_n + \ldots + F_{d_m}(x_1, \ldots, x_{n-1})x_n^{d_n}.$$

At the end of ith iteration we know the values of $F_0(p_1^i, p_2^i, \ldots, p_{n-1}^i)$, $F_1(p_1^i, p_2^i, \ldots, p_{n-1}^i)$, $\ldots$, $F_{d_n}(p_1^i, p_2^i, \ldots, p_{n-1}^i)$. This steps are repeated for $i = 1, q$, where $q = (d_1+1)\ldots(d_{n-1}+1)$. Each of the F_i's is computed using Vandermonde interpolation and has up to q terms. The running time of the resulting algorithm is $O(qp^3 + d_m q^2)$, where $p = \max(m, d_n)$. In the event $d_i = m$, $q = (m+1)^{n-1}$ and the overall running time is $O(m^{n+2} + m^{2n-1})$.

In Macaulay's and Sylvester's formulation of resultants, each entry of the matrix is a linear polynomial in the coefficients of the given system of equations. As a result we express $\mathbf{M}$ as

$$\mathbf{M} = \mathbf{M}_{\mathbf{1}} x_n + \overline{\mathbf{M}},$$

where $\mathbf{M}_{\mathbf{1}}$ is a numeric matrix and entries of $\overline{\mathbf{M}}$ are polynomials in $x_1, \ldots, x_{n-1}$. We either multiply $\overline{\mathbf{M}}$ by $\mathbf{M}_{\mathbf{1}}^{-1}$ or use the transformations presented in the previous section to reduce it to characteristic polynomial computation problem. Since computing the characteristic polynomial involves a constant factor as compared to determinant, we have reduced the symbolic complexity by one variable.

4.4. Probabilistic Interpolation

In this section we combine Zippel's probabilistic algorithm for interpolation with the univariate determinant computation algorithm. The notation used here corresponds to the description of Zippel's algorithm in Section 3.3.

To improve the performance of Zippel's algorithm we use the univariate determinant algorithm at each stage of the algorithm. Let the polynomial obtained after the kth stage of the algorithm be of the form

$$F_k(x_1,\ldots,x_k) = F(x_1,x_2,\ldots,x_k,r_{k+1},\ldots,r_n)$$
$$= c_{1,k}m_{1,k} + \ldots + c_{\alpha,k}m_{\alpha,k},$$

where $r_{k+1},\ldots,r_n$ correspond to the random numbers chosen at the start of the algorithm. At the $k+1$th stage of the algorithm, F_{k+1} is represented as

$$F_{k+1}(x_1,\ldots,x_{k+1}) = F(x_1,\ldots,x_{k+1},r_{k+2},\ldots,r_n)$$
$$= h_1(x_{k+1})m_{1,k} + \ldots + h_\alpha(x_{k+1})m_{\alpha,k},$$

where each $h_i(x_{k+1})$ is a polynomial of degree d_{k+1}. Let $h_i(x_{k+1})$ be a polynomial of the form

$$h_i(x_{k+1}) = hi,0 + h_{i,1}x_{k+1} + \ldots + h_{i,d_k+1}x_{k+1}^{d_k+1}.$$

To compute each $h_i(x_{k+1})$ our algorithm proceeds in the following manner:

- For $1 \le i \le \alpha$ do:
  Compute $F(p_1^i, p_2^i, \ldots, p_k^i, x_{k+1}, r_{k+2}, \ldots, r_n)$ using the univariate determinant algorithm. It corresponds to a polynomial of the form $h_1(x_{k+1})b_{1,k,i} + h_2(x_{k+1})b_{2,k,i} + \ldots + h_{\alpha_k}(x_{k+1})b_{\alpha,k,i}$, where $b_{j,k,i} = m_{j,k_{x_1=p_1^i,x_2=p_2^i,\ldots,x_k=p_k^i}}$.
- For $0 \le j \le d_{k+1}$ do:
  Compute $h_{1,j}, h_{2,j} \ldots, h\alpha, j$ by solving the $\alpha \times \alpha$ Vandermonde system:

$$\begin{pmatrix} b_{1,k,1} & b_{2,k,1} & \ldots & b_{\alpha,k,1} \\ b_{1,k,2} & b_{2,k,2} & \ldots & b_{\alpha,k,2} \\ \vdots & \vdots & \vdots & \vdots \\ b_{1,k,\alpha} & b_{2,k,\alpha} & \ldots & b_{\alpha,k,\alpha} \end{pmatrix}, \begin{pmatrix} h_{1,j} \\ h_{2,j} \\ \vdots \\ h_{\alpha,j} \end{pmatrix} = \begin{pmatrix} c_{1,j} \\ c_{2,j} \\ \vdots \\ c_{\alpha,j} \end{pmatrix},$$

  where $c_{i,j}$ corresponds to the coefficient of x_{k+1}^j in $F(p_1^i, p_2^i, \ldots, p_k^i, x_{k+1}, r_{k+1}, \ldots, r_n)$.
- Substitute $h_i(x_{k+1})$'s to compute F_{k+1}.

The running time of the $k+1$th stage of Zippel's algorithm presented in Section 1.3.3 is $O(\alpha d_{k+1}m^3 + d_{k+1}\alpha^2 + \alpha d_{k+1}^2)$. This has been improved to $O(\alpha p^3 + d_{k+1}\alpha^2)$, where $p = \max(m, d_{k+1})$ by the algorithm presented above.

5. IMPLEMENTATION

In this section, we describe our implementation of these algorithms. An important component of the implementation is the use of modular arithmetic and suitable choice of finite fields.

5.1. Modular Arithmetic

An important problem in the context of finite field computations is getting a tight bound on the coefficients of the resulting polynomial. Since the resultant can always be expressed as a ratio of determinants (or a single determinant), it is possible to use Hadamard's bound for determinants to compute a bound on the coefficients of the resultant [29]. In practice, we found such bounds rather loose and use a randomized version of the Chinese remainder algorithm to compute the actual bignums. The main idea is to compute the coefficients modulo different primes, use the the Chinese remainder theorem for computing the bignum and make a check whether the bignum is the actual coefficient of the determinant. This process stops when the products of all the primes (used for finite field representation) is greater than twice the magnitude of the largest coefficient in the output. It is possible that for a certain choice of primes, the algorithm results in a wrong answer. We show that the probability of failure is bounded by l/p, where l corresponds to the number of primes used in the Chinese remainder algorithm, p is the magnitude of the smallest prime number used in this computation.

The algorithm proceeds in stages. At the k-th stage a random prime number (p_k) is chosen and $G_k(x_1,\ldots,x_n) = F(x_1,\ldots,x_n) \bmod p_k$ is computed using the interpolation algorithm. Let

$$G_k(x_1,\ldots,x_n) = c_{1,k}m_1 + \ldots + c_{q,k}m_q.$$

Thus, the coefficients of various G_i's satisfy the relation

$$c_{i,1} = c_i \bmod p_1$$
$$\vdots$$
$$c_{i,k} = c_i \bmod p_k,$$

These $c_{i,j}$'s are used for computing the bignum, $r_{i,k}$ using the Chinese remainder theorem, and satisfying the relations [29]

$$r_{i,k} \bmod p_j = c_{i,j}, \quad j = 1,k$$

While using the Chinese remainder theorem it is assumed that $r_{i,k}$ are integers lying in the range $(-\frac{p_1p_2\ldots p_k}{2}, \frac{p_1p_2\ldots p_k}{2})$. At this stage we compare the bignums corresponding to the $(k-1)$ and k stages. If

$$r_{i,k-1} = r_{i,k}, \quad i = 1,q$$

it is reasonable to assume that $r_{i,m} = c_i$, $i = 1,q$ and the algorithm terminates. The base case is $k = 2$. Otherwise we repeat the computation for the $(k+1)$ stage.

In general, $l+1$ prime numbers are being used, where l is the minimum integer satisfying the relation

$$2|c_j| < p_1p_2\ldots p_{l+1}$$

and $|c_j|$ corresponds to the coefficient of maximum magnitude of the resultant and p_j's are randomly chosen primes. Thus, the algorithm is:

1. Compute the $c_{i,j}$'s using the probabilistic interpolation algorithm.
2. Given $c_{i,1}, c_{i,2}, \ldots, c_{i,j}$, use the Chinese remainder theorem to compute $r_{i,j}$ for $i = 1, q$.
3. If $r_{i,j-1} = r_{i,j}$ for $i = 1, q$, then $c_i = r_{ij}$, else repeat the steps given above.

The probability of error this algorithm has been analyzed in [36].

5.2. Choice of Finite Fields

The interpolation algorithm based on the randomized version of the Chinese remainder theorem is output sensitive. In other words the time complexity of the algorithm is directly proportional to k, the number of primes used in the finite field computation. To minimize k, we use primes of maximum magnitude possible.

Most of the workstations are 32 bit machines. In other words, all machine instructions for integer arithmetic operate on operands, whose magnitude is bounded by 2^{32}. However, if we use operands of order 2^{32}, simple operations like addition, subtraction can result in overflow and to implement them correctly we need to resort to bignum arithmetic.

In our case, we chose primes of the order 2^{30} (in fact always less than this number) to represent the finite fields. Thus, each operand of the finite field is bounded by 2^{30}. As a result the addition and subtraction operations, $(a + b) \bmod p, (a - b) \bmod p$, work correctly in the hardware implementation and there is no overflow error. However, multiplication can still result in overflow. A simple solution is to use finite fields of order 2^{16}, which slows down the algorithm by almost a factor of two. Depending upon the architecture of the machine on which the algorithm is being implemented, we suggest following techniques for implementing the multiplication subroutine:

- Many workstations provide a hardware implementation of the instruction $(a * b) \bmod p$. There may be no equivalent function defined in the higher level languages. As a result, we recommend an assembly level implementation of this subroutine, which takes as arguments a, b, p and return $(a * b) \bmod p$.
- The instruction presented above is rather sophisticated and as a result is not available on most RISC machines (like Sun-4's). However, most architectures have a multiplication instruction, which takes the operands in two different registers and return the result as a 64 bit number in two different registers. Many machines, like the Sun-3, have a division instruction which assumes that the dividend is in two registers. As a result an instruction like $(a*b) \bmod p$ can be implemented as a combination of multiplication and division instructions. Otherwise, let the result of multiplication be contained in registers $r1$ and $r2$. Thus,

$$a * b = r1 * 2^{32} + r2$$

The fact $a < 2^{30}, b < 2^{30}$ implies $r1 < 2^{28}$. Thus,

$$(a * b) \bmod p =$$

$$((((r1 * 2^{32}) \bmod p) + (r2 \bmod p)) \bmod p =$$

$$(((r1 * c) \bmod p) + (r2 \bmod p)) \bmod p,$$

where $c = 2^{32} \bmod p$ is a precomputed constant. In fact the primes, are chosen such such that c is rather small and the multiplication $(r1 * c)$ does not result in a overflow. Otherwise the multiplication routine is called recursively. Thus, we find that the implementation of the multiplication subroutine uses two multiplication instructions, three remainder instructions (to compute the remainder of integer division) and one addition instruction for 32 bits integers. The actual impact of this multiplication routine on the speed of the overall algorithm is a function of machine's architecture. However, it seems faster (at least on Sun-3 and Sun-4's) than using finite fields of order 2^{16} and for each field we compute the coefficients of the polynomials and bignums using the Chinese remainder theorem.

It is not possible to choose any arbitrary prime for finite field computation. Let $V_2 = (v_{2j})$ represent the elements of the second row of the Vandermonde matrix, as shown in (2), used in the interpolation algorithms. p_k can be used as a prime for finite field computation, if and only if all the elements of the vector $V_{2k} = (v_{ij}) \bmod p_k$ are distinct.

5.3. Choice of Random Numbers

The robustness of the Zippel's probabilistic interpolation algorithm is a function of the random numbers $r_1, \ldots, r_n$ chosen at the first stage of the algorithm. The probability of obtaining an incorrect answer is bounded by nd^2q^2/S (as shown in (3)). We work over finite fields of order $S \approx 2^{30}$. In many applications we expect q to be of the order of 10^4. Moreover, n is at least 3 or 4 and d can be anywhere in the range $(10, 40)$. As a result, the upper bound on the probability of incorrectness is close to one.

The probabilistic version of the Chinese remainder algorithm computes the answer over various finite fields (of order $\approx 2^{30}$). The actual number of finite fields used is a function of the size of the coefficients of the determinant. However, at least two prime fields are used. As a result we choose random numbers $R_i \in (0, 2^{60})$ and at each iteration corresponding to the Chinese remainder theorem, we compute $r_i = R_i \ mod \ p_i$. As a result $S \approx 2^{60}$ and the probability of failure is bounded by 10^{-9} for these applications. Moreover, it is possible to decrease the upper bound on the probability of failure by choosing R_i's from a domain of appropriate size.

5.4. Memory Requirements

The memory requirement of our interpolation algorithm is a linear function of the input and output size. The total amount of space required for resultant computation is $O(m^2 + |c|q)$, where m is the order of the matrix, q is the number of terms in the determinant and $|c|$ correspond to the size of the largest coefficient in the determinant. The algorithm is not constrained by any form of memory requirements, performs efficiently on the workstations and can be even be made to run on personal computers. Moreover given q, it is possible to come up with a tight bound on the running time of the resulting algorithm. The main characteristic of the algorithm is the fact that no intermediate symbolic expressions are generated and the computations are performed over finite fields. This is in contrast to symbolic elimination algorithms based on Gröbner bases, Ritt–Wu's algorithm or resultant algorithms in computer algebra systems, where intermediate symbolic expressions of high complexity are being generated.

6. PERFORMANCE

In this section we compare the performance of different elimination algorithms for implicitizing parametric surfaces. More experiments are planned for other applications. These include comparisons with implementations of Gröbner basis in *Cocoa* and *Macaulay* computer algebra systems.

6.1. Implicitization

We applied the results of this algorithm to implicitizing rational parametric surfaces. Implicitization is a fundamental problem in geometric and solid modeling and also seems to be a standard benchmark for elimination algorithms. Given a parametrization, expressed in projective coordinates,

$$(x, y, z, w) = (X(s,t), Y(s,t), Z(s,t), W(s,t)),$$

we formulate the parametric equations

$$wX(s,t) - xW(s,t) = 0$$

$$wY(s,t) - yW(s,t) = 0$$

$$wZ(s,t) - zW(s,t) = 0$$

and the problem of implicitization corresponds to computing the resultant of these equations by considering them as polynomials in s and t [38]. Other algorithms for implicitization include Gröbner bases and Ritt-Wu's algorithm. Hoffmann has surveyed these techniques in [22]. A particular benchmark for implicitization has been a bicubic parametric surface given by Hoffmann, [20]:

$$\begin{aligned} x &= -3t(t-1)^2 + (s-1)^3 + 3s \\ y &= 3s(s-1)^2 + t^3 + 3t \qquad (8) \\ z &= -3(s(s^2-5s+5)t^3 + (s^3+6s^2-9s+1)t^2 \\ &\quad -(2s^3+3s^2-6s+1)t + s(s-1)). \end{aligned}$$

According to resultant algorithms, the problem of implicitization reduces to expanding a symbolic determinant whose order is equal to the degree of the implicit equation. Furthermore, each entry of the matrix is a linear homogeneous polynomial in x, y, z, w. This is equivalent to a non–homogeneous polynomial in x, y, z. We have highlighted the performance of the multivariate interpolation algorithm presented in Section 3 and an improved version using linear algebra formulations in Section 4. The performance of different algorithms is given in Table 1.

6.2. Implicitizing Parametrizations with Base Points

A base point of a parametrization is the common root of $X(s,t), Y(s,t), Z(s,t), W(s,t)$. They also include the points at infinity. Direct applications of resultants or Gröbner bases fail to implicitize such parametrizations. Modified algorithms using resultants and Gröbner basis are presented by Manocha and Canny, [38] and Kalkbrener, [25], respectively. In particular, the algorithm in [38] considers a perturbed system of the form:

$$wX(s,t) - xW(s,t) = 0$$

$$wY(s,t) - yW(s,t) = 0$$

$$wZ(s,t) - zW(s,t) + \lambda G(s,t) = 0,$$

where $G(s,t)$ is a random polynomial. The resultant is a polynomial of the form $R(x,y,z,\lambda)$. Let us express the resultant as a polynomial in λ and say the coefficient of the lowest degree term is $P(x,y,z)$. We are only interested in $P(x,y,z)$ as opposed to the entire polynomial $R(x,y,z,\lambda)$. $P(x,y,z)$ can be decomposed as $P(x,y,z) = F(x,y,z)G(x,y)$, where $F(x,y,z)$ is the implicit equation and $G(x,y)$ corresponds to the projection of seam curves (images of base points) [38]. Given a prime p, the computation of $P(x,y,z) \bmod p$ using dense interpolation algorithm takes about 1180 sec. on an IBM RS/6000 for a bicubic parametrizatio

Algorithm	*Time*	*Machine*	*Reference*
Gröbner bases	10^5 sec.	Symbolics 3650	Hoffmann [20]
Ritt-Wu's algorithm	28,000 sec.	Sun–3	Gao and Chou [17]
Multipolynomial resultants	138 sec.	IBM RS/6000	[36]
Improved multipolynomial resultants	24 sec.	IBM RS/6000	Section 4

TABLE 1. The performance of different implicitization algorithms on the bicubic parametrization

n with base points [38]. Using the improved interpolation algorithms presented in Section 4, computation of $P(x, y, z) \bmod p$ takes about 58 sec. on IBM RS/6000 for the same parametrization.

7. ACKNOWLEDGEMENTS

The author is grateful to John Canny, James Demmel and Chris Hoffmann for productive discussions on symbolic computations, linear algebra and geometric applications. The author would like to thank the reviewers and the editor for their comments.

REFERENCES

[1] S. S. Abhyankar. Historical ramblings in algebraic geometry and related algebra. *American Mathematical Monthly*, 83:409–448, 1976.

[2] D. N. Bernshtein. The number of roots of a system of equations. *Funktsional'nyi Analiz i Ego Prilozheniya*, 9(3):1–4, 1975.

[3] M. Ben-Or, D. Kozen, and Reif J. The complexity of elementary algebra and geometry. *Journal of Comp. and Sys. Sciences*, 32:251–264, 1986.

[4] M. Ben-or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *ACM Symposium on Theory of Computing*, pages 301–309, 1988.

[5] B. Buchberger. Gröbner bases: An algorithmic method in ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Co., 1985.

[6] B. Buchberger. Applications of Gröbner bases in non–linear computational geometry. In D. Kapur and J. Mundy, editors, *Geometric Reasoning*, pages 415–447. MIT Press, 1989.

[7] T. Beelen and P. Van Dooren. An improved algorithm for the computation of Kronecker's canonical form of a singular pencil. *Lin. Alg. Appl.*, 105:9–65, 1988.

[8] J. Canny. Some algebraic and geometric computations in pspace. In *ACM Symposium on Theory of Computing*, pages 460–467, 1988.

[9] J. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.

[10] J. Canny. Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9:241–250, 1990.

[11] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAECC*, 1993.

[12] J. Canny, E. Kaltofen, and Y. Laksman. Solving system of nonlinear polynomial equations faster. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, 1989.

[13] G.E. Collins. The calculation of multivariate polynomial resultant. *Journal of ACM*, 18(4):515–532, 1971.

[14] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture Notes in Computer Science*, number 33, Springer-Verlag, 1975.

[15] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[16] B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[17] X.S. Gao and S.C. Chou. Implicitization of rational parametric equations. Technical report, Department of Computer Science TR-90-34, University of Texas at Austin, 1990.

[18] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. Discriminants of polynomials in several variables and triangulations of newton polytopes. *Algebra i analiz*, 2:1–62, 1990.

[19] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

[20] C.M. Hoffmann. Algebraic and numeric techniques for offsets and blends. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computations of Curves and Surfaces*, pages 499–528. Kluwer Academic Publishers, 1990.

[21] C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532, 1990.

[22] C. Hoffmann. Implicit curves and surfaces in computer aided geometric design. Technical report, Department of Computer Science CER-92-002, Purdue University, 1992.

[23] D. Ierardi. Quantifier elimination in the theory of algebraically closed fields. In *ACM Symposium on Theory of Computing*, volume 21, pages 138–147, 1989.

[24] Jean-Pierre Jouanolou. *Le Formalisme du Résultant*, volume 90 of *Advances in Mathematics*. 1991.

[25] M. Kalkbrener. *Three Contributions to Elimination Theory*. PhD thesis, Johannes Kepler Universitat, Linz, Austria, 1991.

[26] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoretical Computer Science*, 36:309–317, 1985.

[27] E. Kaltofen and Y.N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *Lecture Notes in Computer Science*, volume 358, pages 467–474. Springer-Verlag, 1988.

[28] D. Kapur and Y.N. Laksman. Elimination methods: An introduction. In D. Kapur B. Donald and J. Mundy, editors, *Symbolic and Numerical Computation: an Integration*. Academic Press, 1992.

[29] D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1981.

[30] R. Loos. Computing rational zeros of integral polynomials by p-adic expansion. *SIAM Journal on Computing*, 7:286–293, 1983.

[31] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[32] F.S. Macaulay. Note on the resultant of a number of polynomials of the same degree. *Proceedings of London Mathematical Society*, pages 14–21, June 1921.

[33] F.S. Macaulay. *The Algebraic Theory of Modular Systems*. Stechert–Hafner Service Agency, New York, 1964.

[34] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[35] F. Morley and A.B. Coble. New results in elimination. *American Journal of Mathematics*, 49:463–488, 1927.

[36] D. Manocha and J. Canny. Efficient techniques for multipolynomial resultant algorithms. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 86–95, 1991.

[37] D. Manocha and J. Canny. Algorithms for implicitizing rational parametric surfaces. *Computer Aided Geometric Design*, 9:25–50, 1992.

[38] D. Manocha and J. Canny. The implicit representation of rational parametric surfaces. *Journal of Symbolic Computation*, 13:485–510, 1992.

[39] D. Manocha and J. Canny. Multipolynomial resultant algorithms and linear algebra. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 158–167, 1992.

[40] F. Morley. The eliminant of a net of curves. *American Journal of Mathematics*, 47:91–97, 1925.

[41] J. Renegar. On the computational complexity and geometry of the first order theory of the reals. Technical Report 854, 856, 858, Cornell University, 1989. Parts I,II,III.

[42] J.F. Ritt. *Differential Algebra*. AMS Colloquium Publications, 1950.

[43] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York, 1885.

[44] B. Sturmfels. Sparse elimination theory. In D. Eisenbud and L. Robbiano, editors, *Computational Algebraic Geometry and Commutative Algebra*. Cambridge University Press, 1991.

[45] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 1992.

[46] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 1948.

[47] P. Van Dooren. The computation of Kronecker's canonical form of a singular pencil. *Lin. Alg. Appl.*, 27:103–141, 1979.

[48] B.L. Van Der Waerden. *Modern Algebra (third edition)*. F. Ungar Publishing Co., New York, 1950.

[49] H.S. White. Bezout's theory of resultants and its influence on geometry. *Bulletin of the American Mathematical Society*, 112:59–68, 1909.

[50] J.H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, Oxford, 1965.

[51] W. Wu. On the decision problem and the mechanization of theorem proving in elementary geometry. *Scientia Sinica*, 21:150–172, 1978. Also in Bledsoe and Loveland, eds. *Theorem Proving: After 25 years, Contemporary Mathematics*, **29**, 213–234.

[52] R. Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9:375–403, 1990.