Probabilistic Algorithms for Computing Resultants

Michael Monagan
Centre for Experimental and Constructive Mathematics
Simon Fraser University, Burnaby, B.C. V5A 1S6, CANADA
mmonagan@cecm.sfu.ca*

ABSTRACT

Let A and B be two polynomials in $\mathbb{Z}[x,y]$ and let $R = \text{res}_x(A,B)$ denote the resultant of A and B taken wrt x. In this paper we modify Collins' modular algorithm for computing R to make it output sensitive. The advantage of our algorithm is that it will be faster when the bounds needed by Collins' algorithm for the coefficients of R and for the degree of R are inaccurate. Our second contribution is an output sensitive modular algorithm for computing the *monic* resultant in $\mathbb{Q}[y]$. The advantage of this algorithm is that it is faster still when the resultant has a large integer content. Both of our algorithms are necessarily probabilistic.

The paper includes a number of resultant problems that motivate the need to consider such algorithms. We have implemented our algorithms in Maple. We have also implemented Collins' algorithm and the subresultant algorithm in Maple for comparison. The timings we obtain demonstrate that a good speedup is obtained.

Categories and Descriptors: I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms – *Algebraic algorithms.*

General Terms: Algorithms

Keywords: Sylvester's resultant, polynomial resultants, modular algorithms, probabilistic algorithms.

1. INTRODUCTION

Let $A = a_m x^m + a_{m-1}x^{m-1} + ... + a_0$ and $B = b_n x^n + b_{n-1}x^{n-1} + ... + b_0$ be two non-zero polynomials in x over a commutative ring $\mathbf{R}$ of degrees m and n respectively. Let $R = \text{res}_x(A,B)$ denote the resultant of A and B taken with respect to x. If $m = 0$ the resultant is a_0^n. If $n = 0$ the resultant is b_0^m. If $m > 0$ and $n > 0$, the resultant is the determinant of Sylvester's matrix. Sylvester's matrix is the following $m+n$ by $m+n$ matrix S over $\mathbf{R}$. The coefficients of A are repeated n times in the first n rows, and the coefficients of B are repeated m times in the last m rows. Thus the resultant is an element of the ring $\mathbf{R}$.

$$S = \begin{bmatrix} a_m & a_{m-1} & ... & a_0 & 0 & ... & 0 \\ 0 & a_m & a_{m-1} & ... & a_0 & ... & 0 \\ \vdots & & \ddots & \ddots & & \ddots & \vdots \\ 0 & ... & 0 & a_m & a_{m-1} & ... & a_0 \\ b_n & b_{n-1} & ... & b_0 & 0 & ... & 0 \\ 0 & b_n & b_{n-1} & ... & b_0 & ... & 0 \\ \vdots & & \ddots & \ddots & & \ddots & \vdots \\ 0 & ... & 0 & b_n & b_{n-1} & ... & b_0 \end{bmatrix}$$

In this paper we are interested in computing the resultant when the coefficients of A and B are polynomials over the integers. In particular we focus on the bivariate case where A and B are elements of $\mathbb{Z}[x,y]$ and $R \in \mathbb{Z}[y]$. Because $\mathbb{Z}[y]$ is an integral domain, the resultant may be computed using fraction-free algorithms. The best fraction-free algorithm is the subresultant algorithm of Brown and Traub [1]. It is a fraction-free modification of Euclid's algorithm. It requires $O(mn)$ arithmetic operations in $\mathbb{Z}[y]$.

For $\mathbb{Z}[y]$, the fastest method is the modular method of Collins [2]. Collins' method computes the resultant modulo a sequence of primes $S = p_1, p_2, p_3, ...$ For each prime $p \in S$ it computes the resultant at $y = \alpha_0, y = \alpha_1, ..., \in \mathbb{Z}_p$. The resultant $R \in \mathbb{Z}[y]$ is reconstructed from these images in $\mathbb{Z}_p$ using polynomial interpolation and Chinese remaindering. This reduces the problem to computing many resultants in $\mathbb{Z}_p[x]$. The following property (see Ch. 9 of [6] for a proof),

$$\text{res}_x(A,B) = (-1)^{mn} b_n^{m-\deg C} \text{res}_x(B,C),$$

where C is the remainder of A divided by B, means we can modify the ordinary Euclidean algorithm to compute the resultant in $\mathbb{Z}_p[x]$ using $O(nm)$ arithmetic operations in $\mathbb{Z}_p$.

Let $R = c_d y^d + c_{d-1}y^{d-1} + ... + c_0$. Let h be the height of R, that is, $h = \max(|c_d|, |c_{d-1}|, ..., |c_0|)$. To apply Collins' method, one needs a degree bound D on the degree of the resultant, i.e., $D \geq d$ and, a coefficient bound H on the height of the resultant, i.e., $H \geq h$. The algorithm will use sufficiently many primes p_i such that $\Pi p_i > 2H$. Here, the factor of 2 is to allow for both positive and negative coefficients in $\mathbb{Z}$. For each prime p it will need $D+1$ evaluation

*This work was supported by the MITACS NCE of Canada and NSERC of Canada

ISSAC'05, July 24–27, 2005, Beijing, China.

points from $\mathbb{Z}_p$. Suppose we use 31 bit primes on a 32 bit machine. Recall that the prime number theorem states that the number of primes $< x$ is asymptotically $x/\ln x$. Thus there are approximately $(2^{31}/31 - 2^{30}/30)/\ln 2 = 48.3$ million 31 bit primes. This means we can reconstruct integers in R of size approximately 1.5 billion bits using 31 bit primes. If we do this then the modular resultant algorithm will use $M = O(\log_{2^{31}} H)$ 31 bit primes.

Suppose the cost of computing one resultant of $A(\alpha_j)$ mod p_i and $B(\alpha_j)$ mod p_i is bounded by C. Then the cost of Collins' modular resultant algorithm, assuming classical (i.e. quadratic) algorithms for interpolation and Chinese remaindering, is $O(CM(D+1)) + O(M(D+1)^2) + O(M^2(D+1)) = O(MD(C+M+D))$. These three contributions are for the $M(D+1)$ modular resultants, M interpolations of $D+1$ points, and applying the Chinese remainder theorem to at most $D+1$ coefficients, respectively. Observe that any over estimate of either the degree bound or the coefficient bound will affect the cost of the algorithm *proportionately*. Thus for a practical implementation of Collin's algorithm, we need to consider how to obtain good bounds.

One can obtain a bound on $d = \deg_y R$ from Sylvester's matrix using the rows or the columns. That is,

$$D_{row} = \sum_{i=1}^{m+n} \max_{j=1}^{m+n} \deg_y S_{i,j} \text{ and } D_{col} = \sum_{j=1}^{m+n} \max_{i=1}^{n+m} \deg_y S_{i,j}$$

both bound d. We also have the Bezout theorem which says that the number of roots of $R(y)$ is bounded by $D_{bez} = \deg A \times \deg B$ where $\deg A$ denotes the total degree of the polynomial A in x and y. The D_{bez} bound is better than D_{col} and D_{row} when the inputs are dense. In our experience, the bound D_{col} based on the columns of S is usually better, often by a factor of two than D_{row}, though the row bound can be better, e.g., if $m \gg n$. We compute all three and take $D = \min(D_{row}, D_{col}, D_{bez})$.

Let S' be the matrix of integers where $S'_{i,j}$ is the one-norm of $S_{i,j}$, i.e., the sum of the absolute value of the coefficients of $S_{i,j}$. In [7], Goldstein and Graham prove that Hadamard's bound on $\det(S')$ bounds the height of the coefficients of $\det(S)$. Hadamards bound is $\Pi_{i=1}^{m+n} \sqrt{\sum_{j=1}^{m+n} {S'_{i,j}}^2}$. One can compute Hadamard's bound along the rows or down the columns of S' and use the smaller of the two bounds. Again, we find that the bound obtained using the columns of S' is usually better than that obtained using the rows.

How good are these bounds? If one fixes the degree in y of all coefficients of A and B in x and chooses random integers for the integer coefficients of A and B from a sufficiently large set, say 10 digit random integers, then the degree bound will be exact and the coefficient bound will be almost tight. If the bounds for h and d could off by a factor of 2 at most, then we would be happy. It turns out, however, that there are real examples where these bounds can be arbitrarily far off. In those cases, the subresultant algorithm can be better, sometimes much better, than the modular resultant algorithm. This makes algorithm selection difficult. To motivate the need for an improved modular algorithm, we need to study some real problems. We will consider four problems in section 2. In section 3 and 4 we give two new probabilistic algorithms. We end the introduction with a complete description of the modular resultant algorithm of Collins [2] for inputs $A, B \in \mathbb{Z}[x, y]$.

Collins' Algorithm

In the specification of the algorithm, $R = \text{res}_x(A, B)$. So $R = 0$ or $R = c_d y^d + c_{d-1} y^{d-1} + ... + c_{l+1} y^{l+1} + c_l y^l$ for some $d \geq l \geq 0$ and $c_d c_l \neq 0$. We call l the *low degree* of the resultant and c_l the trailing coefficient. The algorithm assumes a degree bound $D \geq d$ and also a low degree bound $0 \leq L \leq l$. A low degree bound L can also be obtained directly from Sylvester's matrix. In our presentation of Collins' algorithm we improve the efficiency when a nontrivial low degree bound L is known. For if $R(y)$ is divisible by y^L then $D - L + 1$ evaluation points are sufficient.

Definition 1. *Let $A = a_m x^m + ... + a_0$ and $B = b_n x^n + ... + b_0$ be non-zero polynomials in $\mathbb{Z}[y][x]$ of degrees m and n. A prime p is said to be* bad *if $a_m \equiv 0 \bmod p$ or $b_n \equiv 0 \bmod p$. Similarly, an evaluation point $\alpha \in \mathbb{Z}_p$ is said to be* bad *if $a_m(\alpha) \equiv 0 \bmod p$ or $b_n(\alpha) \equiv 0 \bmod p$.*

The correctness of the algorithm follows from noting that if p is not a bad prime and α is not a bad evaluation, then

$$R(\alpha) \bmod p = \text{res}_x(A(\alpha) \bmod p, B(\alpha) \bmod p).$$

Algorithm CRES

Input $A, B \in \mathbb{Z}[x, y] \backslash \{0\}$ of degree m and n resp.
Input $D \geq L \geq 0$ satisfying $D \geq d$ and $L \leq l$.
Input $H \geq h = \max(1, |c_d|, |c_{d-1}|, ..., |c_{l+1}|, |c_l|)$.
Output $R = \text{res}_x(A, B) \in \mathbb{Z}[y]$.

1 Initialize $M = 1, \Delta = D - L$.

2 Initialize S to the set of primes such that $\Pi_{p \in S}(p) > 2H$, and, for each prime $p \in S$, $p > \Delta + \deg_y A + \deg_y B$ and p is not a bad prime.

3 REPEAT

- 3.1 Choose the next prime p from S. Set $A_p = A \bmod p$ and $B_p = B \bmod p$.
- 3.2 Set $N = \Delta$ and choose $N+1$ distinct non-zero evaluation points $\alpha_0, \alpha_1, ..., \alpha_N$ from $\mathbb{Z}_p$ such that $a_m(\alpha_i) \not\equiv 0 \bmod p$ and $b_m(\alpha_i) \not\equiv 0 \bmod p$.
- 3.3 FOR $i = 0, 1, ..., N$ DO
  Compute $r_i \in \mathbb{Z}_p$ the resultant of $A_p(\alpha_i)$ and $B_p(\alpha_i)$ modulo p using the Euclidean algorithm and set $r_i = r_i/\alpha_i^L \bmod p$.
- 3.4 Interpolate $r \in \mathbb{Z}_p[y]$ from (α_i, r_i) for $i = 0..N$. Set $r = y^L r$.
- 3.5 IF $M = 1$ then set $M = p, \bar{R} = r$.
- 3.6 ELSE (apply the Chinese remainder theorem) Solve $C \equiv r \bmod p$ and $C \equiv \bar{R} \bmod M$ for C. Set $M = pM, \bar{R} = C$.

UNTIL $M > 2H$.

4 Put the coefficients of $\bar{R}$ in the symmetric range for $\mathbb{Z}_M$. Output $\bar{R}$.

Remark: We require that $p > \Delta$ so that there are sufficiently many points for interpolating the resultant in $\mathbb{Z}_p[y]$. The requirement that $p > \Delta + \deg_y a_m + \deg_y b_n$ allows for the possibility that there could be as many as $\deg_y a_m + \deg_y b_n$ bad evaluation points which must be avoided.

Remark: The set of primes S is usually taken to be the biggest primes which the hardware of the machine supports. Thus if we are using 31 bit primes on a 32 bit machine, we would start with the largest such prime and simply count down towards 0 skipping primes which are bad.

Remark: If y divides neither A nor B, the low degree bound L computed from the rows of Sylvester's matrix must be 0, hence, a non-trivial bound for l can only come from the columns of Sylvester's matrix. If y divides A or B then one should apply $\mathrm{res}_x(A = y^i\bar{A}, B = y^j\bar{B}) = y^{ni+mj}\mathrm{res}_x(\bar{A}, \bar{B})$.

2. HOW GOOD ARE THE BOUNDS?

We first look at four resultant problems to investigate how good the degree bounds and height bounds for the resultant are. If they are not good then the obvious line of attack would be to look for better bounds. The examples, however, tell us that it will be impossible to obtain accurate bounds for real problems in general.

Example 2.1 (Cyclotomic Polynomials)
Let $\Phi_n(x) \in \mathbb{Z}[x]$ denote the n'th cyclotomic polynomial. For $2 \le i < j \le 200$ we find that if i does not divide j then the resultant of $\Phi_i(x)$ and $\Phi_j(x)$ is 1! Let $S_{i,j}$ be the Sylvester matrix for $\Phi_i(x)$ and $\Phi_j(x)$. Hadamard's bound for $S_{i,j}$ is a moderately large integer. For $i = 197$ and $j = 199$ Hadamard's bound is $> 10^{427}$ using the columns of $S_{i,j}$ and $> 10^{452}$ using the rows of $S_{i,j}$. This example shows that the height bound may be arbitrarily far off. It tells us also that it will be impossible to obtain accurate bounds in general without actually computing R. Furthermore, for $i = 197$, $j = 199$, the remainder of $\Phi_j(x)$ divided $\Phi_i(x)$ is $x + 1$. This means the subresultant algorithm will be much faster than Collins' modular resultant algorithm.

Example 2.2 (The Logistic Map)
Let $f(x) = ax(1-x)$ for parameter a. Let $g(x) = f^{(n)}(x)$. The bifurcation points of f are roots of the polynomial $r = \mathrm{res}_x(g(x) - x, g'(x) + 1) \in \mathbb{Z}[a]$. For example, for $n = 2$,

$$\begin{aligned} g(x) - x &= f(f(x)) - x \\ &= -x(ax + 1 - a)(a^2x^2 - a^2x - ax + a + 1), \\ g'(x) + 1 &= 1 + (1 - 2x)a^2 + (-4x^3 + 6x^2 - 2x)a^3. \end{aligned}$$

The resultant

$$r = -a^9(a^2 + 1)(a^2 - 4a + 5)(a^2 - 2a - 5)^2.$$

Notice the high power of a dividing the resultant even though neither input is divisible by a. In the following table we compare the bounds for the degrees and heights with the actual values for $n = 3, 4, 5, 6, 7, 8$. Columns l and d are the actual low degree and degree of r in a. Columns L and D are the best low degree and degree bounds for a. Column m is the number of decimal digits of the height h of r and column M is the number of digits of the bound H for the height of r.

n	L	D	l	d	M	m
3	37	98	49	73	24	7
4	148	430	225	289	97	19
5	571	1798	961	1121	383	48
6	2202	7350	3969	4353	1501	115
7	8569	29718	16129	17025	5894	269
8	33664	119510	65033	67073	23262	615

Observe that for $n = 8$ the height bound is off by a factor of over 38, and the number of points necessary to interpolate $r(a)/a^l$, namely $d - l + 1$, is 42 times less than the degree bounds $D - L + 1$ yield. Thus if we knew l, d and m the modular algorithm would run over 1000 times faster!

These examples motivate us to design an output sensitive modular resultant algorithm which will be probabilistic, that is, it will output R with controllable high probability, and, the number of primes and number of evaluation points used depends on the size of the resultant R. We detail this algorithm, algorithm PRES, in section 2. We have implemented it in Maple [9]. Below is a timing comparison comparing it with Collins' algorithm and the subresultant algorithm.

n	Subres	Collins	PRES	factor
3	0.01	.02	.01	0.01
4	0.49	.32	.06	0.01
5	21.3	13.2	.57	0.05
6	1359	688.8	7.6	0.20
7	143,514	41,354	120	2.05
8	–	–	2,092	19.9

The timing data (in CPU seconds) was obtained using a 64 bit compile of Maple on an AMD Opteron 248 processor running at 2.2 GHz. On such a machine, Maple uses 31.5 bit primes. We see that Collins' algorithm is not much faster than the subresultant algorithm. Algorithm PRES is over 300 times faster than Collins' algorithm for $n = 7$. The timings in the last column are for factoring the resultant. They are included to emphasize that the hard part of this problem is computing the resultant.

Example 2.3 (Primary Decomposition of Ideals)
Let I be a zero-dimensional ideal in $\mathbb{Q}[x, y, z, ...]$. One way to compute the primary decomposition of I is to first compute a Gröbner basis G for I using a lex ordering. Suppose the Gröbner basis is of the form

$$G = \{g_1(x), g_2(x, y), g_3(x, y, z), ...\}.$$

One would then factor $g_1 \in \mathbb{Q}[x]$. Suppose $g_1(x)$ is irreducible and g_2 is not linear in y. One would then factor g_2 modulo g_1. To factor g_2 modulo g_1 one may use Trager's algorithm [12]. Trager's algorithm will begin by factoring the polynomial $r(y) = \mathrm{res}_x(g_1(x), g_2(x, y)) \in \mathbb{Z}[y]$. If $r(y)$ is square-free, then, for each irreducible factor f_i of r, Trager's algorithm computes $h_i = \gcd(f_i, g_2)$ modulo g_1, an irreducible factor of g_2.

Consider the following example from the POSSO test suite [11] where $I \subset \mathbb{Q}[x, y, z, t, u]$ is generated by

$$\begin{aligned} &-1 + 2x^2 - 2y^2 + 2z^2 - 2t^2 + 2u^2, \\ &-1 + 2x^3 - 2y^3 + 2z^3 - 2t^3 + 2u^3, \\ &-1 + 2x^4 - 2y^4 + 2z^4 - 2t^4 + 2u^4, \\ &-1 + 2x^5 - 2y^5 + 2z^5 - 2t^5 + 2u^5, \\ &-1 + 2x^6 - 2y^6 + 2z^6 - 2t^6 + 2u^6. \end{aligned}$$

We find that with $u < t < x < y < z$, $G = \{g_1(u),\ g_2(u, t),\ g_3(z, u),\ g_4(y, t, u),\ g_5(x, z, u)\}$. $g_1(u)$ is irreducible of degree 36 with 14 digit coefficients, $g_2(u, t)$ is quadratic in t, of degree 35 in u with 112 digit coefficients, $g_3(u, z)$ is quadratic in z with 113 digit coefficients, $g_4(y, t, u)$ is linear in y and t and $g_5(x, z, u)$ is linear in x and z. The resultant $r = \mathrm{res}_u(g_1, g_2)$ has degree 72 in t and 3966 digit coefficients.

The bounds are good. The bound D on the degree is 72 and the bound H on the coefficients of r is a 4485 digit integer. However, $r = cr'$ where c is a 3931 digit integer content and r' is a polynomial of degree 72 with 36 digit coefficients! Similarly, $\mathrm{res}_u(g_1, g_3)$ also has a large integer content.

In the context of Trager's algorithm, all we need is r'. This motivates us to consider a modular resultant algorithm that computes r' not r. We do this via computing the monic resultant $\tilde{r} = r/\mathrm{lc}_y r(y)$ using a modular algorithm which uses rational number reconstruction [13]. The algorithm, algorithm MRES, is presented in section 4. On this example, we find that the time to compute r using Collins' algorithm is 7.5s and the time to compute r' using algorithm MRES is 0.16s for a speedup of a factor of 47. More significantly, the time Maple takes to compute the entire primary decomposition of I is reduced from 39.1s to 1.7s!

The following example is from Kotsireas [4]. Let $I \subset \mathbb{Q}[t, u, v, x, y, z]$ be generated by

$$(t-u)(x-y) - 2z + 2,$$
$$(t-u)(x+y-2z) - 2(x-y),$$
$$(t-u)(t-v) - 2t - 2u + v + 1,$$
$$x^2t^3 - 1, \quad y^2u^3 - 1, \quad \text{and} \quad z^2v^3 - 1.$$

With $u < v < x < y < t$, the lex Gröbner basis has two polynomials $g_1(u)$ and $g_2(v, u)$ where $g_1(u)$ is of degree 51 in u with 18 digit coefficients and $g_2(v, u)$ is of degree 2 in u with 446 digit coefficients. On this example $r = \mathrm{res}_u(g_1, g_2)$ has degree 102 in v with 22665 digit coefficients. Again, the bounds are good. The bound $D = 102$ and the bound H on the coefficients of R is 23641 digits. Again $r = cr'$ where c is a large integer, a 23607 digit integer and r' has 33 digit coefficients. We find that the time using our Maple implementation of Collins' algorithm is 219.6s, algorithm MRES takes 0.31s for a gain of a factor of 708.

Example 2.4 (SYZYGY Polynomials)
For positive integers m, n, the parametrization of the syzygy figures in $\mathbb{R}^2$ is given by

$$x(t) = \sin(mt) \quad \text{and} \quad y(t) = \cos(nt).$$

We can construct the equation $f(x, y) = 0$ for the curve by first expanding multiple angles using

$$\sin(2t) = 2\sin(t)\cos(t) \quad \text{and} \quad \cos(2t) = 2\cos(t)^2 - 1,$$

applying the rational parametrization

$$\sin(t) = 2t/(1+t^2), \quad \cos(t) = (1-t^2)/(1+t^2),$$

and then computing the resultant

$$f = \mathrm{res}_t((1+t^2)^m(x - \sin(mt)), (1+t^2)^n(y - \cos(nt))$$

in $\mathbb{Z}[x,y]$. For $x(t) = \sin(3t), y(t) = \cos(5t)$ we obtain

$$f(x, y) = 70368744177664(256\,x^{10} - 640\,x^8 + 560\,x^6 + 16\,y^6 - 200\,x^4 - 24\,y^4 + 25\,x^2 + 9\,y^2 - 1)$$

Observe that $f(x, y)$ has a large integer content relative to the size of the height of the primitive part of $f(x, y)$. Observe also that $f(x, y)$ is sparse, the monomials are even powers in x or in y only. We compute the following data for selected m and n. Column D is the degree bounds in x and y for the resultant. Column d is the actual degrees in x and y of the resultant. Column M is the length of the height bound on the resultant, column m is the length of the actual height of $f(x, y)$, column c is the length of the integer content of $f(x, y)$ and column h is the length of the height of the primitive part of $f(x, y)$.

m	n	D	d	M	m	c	h	N
11	13	(26,22)	(26,22)	276	168	159	9	25
11	15	(30,22)	(30,22)	346	195	184	11	27
11	17	(34,22)	(34,22)	416	221	209	12	29
11	19	(38,22)	(38,22)	466	249	235	14	31
11	21	(42,22)	(42,22)	516	275	260	15	33
11	23	(46,22)	(46,22)	566	302	295	17	35
13	15	(30,26)	(30,26)	381	230	219	11	29
15	17	(34,30)	(34,30)	501	301	289	12	33
17	19	(38,34)	(38,34)	637	382	368	14	37
19	21	(42,38)	(42,38)	789	472	457	15	41
21	23	(46,42)	(46,42)	958	583	556	17	45
23	25	(50,46)	(50,46)	1144	883	665	18	49

Observe that the degree bounds are accurate. Observe that the height of the resultant $f(x, y)$ is a factor of 15 to 40 times longer than the height of the primitive part of the resultant. This is another example where a monic resultant algorithm that is output sensitive will help. The last column N shows the number of terms in the resultant $f(x, y)$ indicting how sparse it is. We infer $N = 1 + m + n = 1 + (\deg_x f + \deg_y f)/2$.

3. A PROBABILISTIC ALGORITHM

Algorithm PRES below chooses a prime p from a suitably large set of primes S. It then computes the first image $r_1 \in \mathbb{Z}_p[y]$ using the degree bounds D and L as in algorithm CRES. Thus this first image uses $D - L + 1$ evaluation points. Let $d_1 = \deg_y r_1$ and l_1 be the low degree of r_1 in y. To make the algorithm output sensitive, one could assume $d_1 = d$ and $l_1 = l$ and proceed to use $d_1 - l_1 + 1$ evaluation points for the subsequent primes and terminate when the result of the Chinese remaindering does not change for several iterations, say $K = 10$ iterations. This will not work if $d_1 < d$ or $l_1 > l$. We begin with a definition.

Definition 2. *Let $R = c_d y^d + \ldots + c_l y^l$ where $d \geq l \geq 0$ and $c_d c_l \neq 0$. A prime p is said to be* unlucky *if $p | c_d$ or $p | c_l$.*

Observe that for a given input A and B, the number of unlucky primes is finite.

Example 3.1 (Unlucky Primes)
Consider $A = x^4 + a_3 y^4 x^2 + a_0 y^2$ and $B = x - 1$ in $\mathbb{Z}[y][x]$. Then $R = \mathrm{res}_x(A, B) = a_3 y^4 + a_0 y^2 + 1$. Thus any prime dividing a_3 is unlucky.

Observe that the leading coefficient of R in example 3.1 does not depend on the leading coefficients of A and B. In general, it will not be possible to efficiently compute c_d or c_l from A and B to detect unlucky primes. Thus unlike bad primes, we cannot efficiently avoid unlucky primes in advance.

Suppose the first prime p_1 is unlucky. Our idea to detect this is as follows. When we compute the second image, r_2, the resultant of A and B modulo p_2, instead of using $d_1 - l_1 + 1$ evaluation points, we will use $\delta > 0$ additional evaluation points. If $d_1 = d$ and $l_1 = l$ then when we interpolate the resultant modulo p_2, it must have degree $d_2 \leq d$ and $l_2 \geq l$ where d_2 and l_2 are the degree and low degree of the second image. If $d_1 < d$ or $l_1 > l$ then when we interpolate r_2, provided p_2 is not also unlucky, r_2 will *probably* have degree $d_2 = d_1 + \delta$. If $d_2 > d_1$ then we detect that p_1 was unlucky and we will restart the algorithm with a new prime. By requiring that the algorithm needs K primes of agreement before it can terminate, there are at least K chances that the algorithm can identify an unlucky prime in this way. The timings for algorithm PRES reported in section 2 assumed 31.5 bit primes on a 64 bit machine, $K = 5$, and $\delta = 1$. We present the algorithm.

Algorithm PRES

Input $A, B \in \mathbb{Z}[x,y]\backslash\{0\}$ of degree m and n resp.
Input $D \geq L \geq 0$ satisfying $D \geq d$ and $L \leq l$.
Input $H \geq h = \max(1, |c_d|, |c_{d-1}|, ..., |c_{l+1}|, |c_l|)$.
Input $K \geq 1$ (number of primes of agreement).
Output $R = \mathrm{res}_x(A, B) \in \mathbb{Z}[y]$.

1 Initialize $M = 1, \Delta = D - L, \delta = 0, l = L, d = D$.

2 Initialize S to a set of primes such that $\Pi_{p\in S} p \gg 2H$, and for each $p \in S$, we have $p > \Delta + \deg_y A + \deg_y B$ and p is not a bad prime.

3 REPEAT

3.1 Choose a new prime p from S.

3.2 Set $N = \Delta+\delta$ and choose $N+1$ distinct non-zero evaluation points $\alpha_0, \alpha_1, ..., \alpha_N$ from $\mathbb{Z}_p$ at random such that $a_m(\alpha_i) \not\equiv 0 \bmod p$ and $b_m(\alpha_i) \not\equiv 0 \bmod p$.

3.3 FOR $i = 0, 1, ..., N$ DO

Compute $r_i \in \mathbb{Z}_p$ the resultant of $A(\alpha_i)$ and $B(\alpha_i)$ modulo p using the Euclidean algorithm and set $r_i = r_i/\alpha_i^l \bmod p$.

3.4 Interpolate $r \in \mathbb{Z}_p[y]$ from (α_i, r_i) for $i = 0..N$. Set $r = y^l r$.

3.5 IF $\delta = 0$ set the bounds:

3.5.1 If $r = 0$ set $d = L - 1, l = L$ otherwise set $d = \deg_y(r)$ and l to the low-degree of r in y.

3.5.2 Set $\delta = 1$, $\Delta = d - l$, $M = p$, $\bar{R} = r$, $j = 0$.

3.6 ELIF $\deg_y(r) > d$ then – restart the algorithm Initialize $M = 1, \Delta = D - L, \delta = 0, l = L, d = D$.

3.7 ELSE apply the Chinese remainder theorem:

3.7.1 Solve $C \equiv r \bmod p$ and $C \equiv \bar{R} \bmod M$ for C in the symmetric range.

3.7.2 If $C = \bar{R}$ then set $j = j + 1, M = p \times M$ ELSE set $j = 0, M = p \times M, \bar{R} = C$.

UNTIL $j = K$.

4 Output $\bar{R}$.

If the algorithm outputs $\bar{R} \neq R$ we say that the algorithm "fails". We give two examples which illustrate the two ways in which an adversary, who knows the sequence of primes $p_1, p_2, p_3, ...$ can make the algorithm fail.

Example 3.2 (Likely Failure Case).
Consider $A = x^4+ay^2x^2+y^4$ and $B = x-1$ in $\mathbb{Z}[y][x]$ where $a = p_1p_2 \times ... \times p_{11}$. The resultant $R = \mathrm{res}_x(A, B) = y^4 + ay^2 + 1$. However, if we call algorithm PRES with $K = 10$ then the algorithm will output $y^4 + 1$. The prime p_1 is not unlucky. The algorithm fails because it stabilizes too early.

Example 3.3 (Unlucky Prime Failure).
Consider $A = x^4 + ay^4x^2 + y^2$ and $B = x - 1$ in $\mathbb{Z}[y][x]$ where $a = p_1p_2 \times ... \times p_{11}$. The resultant $R = \mathrm{res}_x(A, B) = ay^4 + y^2 + 1$. If we call algorithm PRES with $K = 10$ then it outputs $y^2 + 1$. The algorithm stabilizes before detecting that p_1 is unlucky.

First we argue that the algorithm must terminate in finite time. If the first prime p_1 is not unlucky, it must terminate K primes after $M > 2h$ where h is the height of R. It could terminate earlier as in example 3.2 with an incorrect output. If p_1 is unlucky then the algorithm must either terminate early as in example 3.3 or it must eventually detect that the first prime was unlucky and restart. It may restart with another unlucky prime. But it must either terminate with an incorrect output or eventually restart with a prime which is not unlucky since the number of unlucky primes is finite.

There are two useful measures for the probability that algorithm PRES fails. The first assumes the coefficients of R modulo a prime p are uniformly distributed on $[0, p)$. This assumption will be true asymptotically over the set of all inputs A, B of bounded size. The second is an adversarial approach. Allow an adversary to choose R to maximize the probability of failure. In both cases we bound the probability of failure and then choose S, K and δ so that this probability of failure is low. The first measure is optimistic, the second pessimistic. The performance of the algorithm on real data will lie between the two. First note that if $R = 0$ the algorithm always outputs 0. Thus from now on assume $R = c_dy^d + ... + c_ly^l$ with $c_dc_l \neq 0$.

Unlucky primes: Let $p_1, p_2, ...$ be the sequence of primes chosen in step 3.1. Assuming that the coefficients of R modulo p are uniformly distributed on $\mathbb{Z}_p$, the $\Pr(p_1|c_d) = 1/p_1$ and $\Pr(p_1|c_l) = 1/p_1$, hence, the $\Pr(p_1 \text{ is unlucky}) < 2/p_1$.

Premature termination with a wrong output: Suppose p_1 is *not* unlucky. Then for each subsequent prime p, the algorithm uses sufficient points to interpolate R modulo p thus at the end of each iteration of step 3 we have $R \equiv \bar{R} \bmod M$. The algorithm can output an incorrect answer if at some iteration j before $\bar{R} = R$, we have $\bar{R} \equiv R \bmod p$ for K consecutive primes $p_{j+1}, p_{j+2}, ..., p_{j+K}$. That is, at the end of iteration $j + K$ we have $P|\bar{R} - R$ where $P = p_{j+1} \times p_{j+2} \times ... \times p_{j+K}$. If $\bar{R} \neq R$ then $\bar{R}$ differs from R in at least one coefficient, say $\bar{c}_i \neq c_i$. Again, assuming the coefficients of R modulo p are uniformly distributed on $\mathbb{Z}_p$, the $\Pr(\bar{c}_i \equiv c_i \bmod P) = 1/P$. But the algorithm could terminate prematurely at any iteration $j > K$ for this reason. Suppose the algorithm uses 31 bit primes. Then it will need no more than $\lceil \log_{2^{31}} h \rceil$ 31 bit primes to reconstruct R. Hence the expected probability of failure when p_1 is not unlucky is $< \lceil \log_{2^{31}} h \rceil / 2^{31K}$.

Remark: In the above argument, if the algorithm fails then $P|\bar{R} - R$, that is, P must divide $\bar{c}_i - c_i$ for all $i = l, l + 1, ..., d$. If R has $d-l+1$ non-zero coefficients then one may be tempted to argue that the $\Pr(P|\bar{R} - R) < 1/P^{d-l+1}$. However this assumes that the coefficients c_i are of the same length. In practice, it is often the case that the coefficients

$c_l, c_{l+1}, ..., c_d$ are of quite different lengths. In particular, c_l, the trailing coefficient of R, is often the largest.

Worst case bound: An adversary can, however, construct inputs such that the resultant $R = c_d y^d + ... + c_l y^l$ with $c_d = p_1, p_3, ...$ and $c_l = p_2, p_4, ...$ where the primes $p_1, p_2, ...$ are chosen from S. On such inputs the probability of hitting unlucky primes is high. Similarly, an adversary can construct inputs such that the resultant $R = c_d y^d + ... + c_i y^i + ... + c_l y^l$ with $c_i = p_1 p_2 p_3 \times ...$ as in example 3.2. In this case, even if p_1 is not unlucky, the probability of terminating too early is higher. The solution to both problems is to choose S so that the probability that any coefficient of R vanishes is low. Construct S, a set of primes, satisfying the conditions in step 2 with $\Pi_{p \in S} p > H^4$ where H is the height bound. Now, if the algorithm chooses primes *at random* from S, then

$$\Pr(\ c_i \equiv 0 \bmod p\ ) < 1/4$$

and

$$\Pr(\ p \text{ is unlucky }) = \Pr(\ p | c_d \text{ or } p | c_l\ ) < 1/2.$$

To make the probability that that p is unlucky less than 2^k we much choose S with $\Pi_{p \in S} p > H^{4k}$.

Unlucky Prime Detection

We now consider the case where p_1 is unlucky. The algorithm computes at least K more images and thus makes at least K attempts to detect if p_1 is unlucky. We will bound the expected probability that the algorithm will fail to discover that p_1 is unlucky in K iterations assuming $\delta = 1$ additional evaluation points are used. We will assume, conservatively, that if a subsequent prime p is unlucky, the algorithm will fail to detect that p_1 is unlucky. Thus is the cases considered below, we assume that the K subsequent primes are not unlucky.

CASE $r_1 = 0$, that is, $p_1 | R(y)$.
Step 3.5.1 sets $\Delta = -1$ so that $N = \Delta + \delta = 0$, hence, the algorithm uses one non-zero evaluation point α_0 to interpolate $R(y)/y^L$. It fails to identify p_1 is unlucky if and only if $R(\alpha_0)/\alpha_0^L \equiv 0 \bmod p$, that is, if and only if α_0 is a root of $R(y)/y^l = c_d y^{d-l} + ... + c_l$. On average, a polynomial of degree n over $\mathbb{Z}_p$ has exactly one root. The worst case occurs when $R(y) = y^n - 1$ and $n | p - 1$ which has exactly n distinct roots. Thus we have

$$\Pr(R(\alpha_0) = 0) \leq (d - l)/(p - 1) \leq (D - L)/(p - 1).$$

CASE $l_1 = d_1 = 0$, that is, r_1 is a constant.
Step 3.5.1 sets $\Delta = 0$ so that $N = \Delta + \delta = 1$, hence, the algorithm uses two non-zero evaluation points α_0 and α_1 to interpolate $R(y)$ in step 3.4. Let $r = a + b(y - \alpha_0)$ where $a = \alpha_0$ and $b = (R(\alpha_1) - R(\alpha_0))/(\alpha_1 - \alpha_0)$. The algorithm fails to identify p_1 is unlucky if and only if $b = 0$, that is, $R(\alpha_1) \equiv R(\alpha_0) \bmod p$. We claim that for $0 < \alpha_0 \neq \alpha_1 < p$, the $\Pr(R(\alpha_0) \equiv R(\alpha_1) \pmod p) < d/p$ where $d = \deg_y(R)$. To show this let

$$c_v = |\{\alpha \in \mathbb{Z}_p, \alpha \neq 0 : R(\alpha) \equiv v \pmod p\}|.$$

Now $\sum c_v = p - 1$ and $0 \leq c_v \leq d$. Because $0 < \alpha_0 \neq \alpha_1 < p$ are chosen at random, we have

$$\begin{aligned}
&\Pr(R(\alpha_0) = R(\alpha_1)) \\
&\quad = \sum_{v \in \mathbb{Z}_p} \Pr(R(\alpha_0) = v) \text{ and } \Pr(R(\alpha_1) = v) \\
&\quad = \sum_{v \in \mathbb{Z}_p} \frac{c_v^2}{(p-1)^2} - \frac{1}{p-1}
\end{aligned}$$

where $1/(p - 1)$ accounts for $\alpha_0 \neq \alpha_1$. This probability is maximized when the c_vs are maximized. Thus since $0 \leq c_v \leq d$ it is

$$\leq \frac{d^2}{(p-1)^2} . \frac{p-1}{d} - \frac{1}{p-1} < \frac{d-1}{p-1} < \frac{d}{p}.$$

This maximum is achieved with $R(y) = y^d - 1$ and $d | (p - 1)$. To summarize, if $R(y) = p_1(y^d - 1)$ and $d | (p - 1)$, p_1 is unlucky and the probability that algorithm PRES fails to detect this is less than d/p.

CASE $d_1 = 1, l_1 = 0$, that is, r_1 is linear and $l = 0$.
Step 3.5.1 sets $\Delta = 1$ so that $N = \Delta + \delta = 2$, hence, the algorithm uses three non-zero evaluation points α_0, α_1 and α_2 to interpolate $R(y)$ in step 3.4. Let

$$r = a + b(y - \alpha_0) + c(y - \alpha_0)(y - \alpha_1)$$

where $a = R(\alpha_0)$ and $b = (R(\alpha_1) - R(\alpha_0))/(\alpha_1 - \alpha_0)$ and

$$c = \frac{R(\alpha_2) - a - b(\alpha_2 - \alpha_0)}{(\alpha_2 - \alpha_0)(\alpha_2 - \alpha_1)}.$$

It fails to identify p_1 is unlucky if and only if $c = 0$, that is,

$$R(\alpha_2) - R(\alpha_0) = \frac{\alpha_2 - \alpha_0}{\alpha_1 - \alpha_0}(R(\alpha_1) - R(\alpha_0)).$$

Because α_0, α_1 and α_2 are non-zero, distinct, and random, the fraction $(\alpha_2 - \alpha_0)/(\alpha_1 - \alpha_0)$ is uniformly distributed on $[2, p - 1]$. If $R(\alpha_2) = R(\alpha_0)$ then the algorithm fails to identify p_1 is unlucky iff also $R(\alpha_1) = R(\alpha_0)$. This happens with probability at most $d^2/(p - 1)^2$. If $R(\alpha_2) \neq R(\alpha_0)$ then if $R(\alpha_1) = R(\alpha_0)$ then the algorithm identifies p_1 is unlucky with probability 1, otherwise it fails to identify p_1 is unlucky with probability at most $1/(p-2)$. Thus the total probability of failure is at most $d^2/(p - 1)^2 + 1/(p - 2)$.

CASES $d_1 = d - 1, l_1 = l$ and $d_1 = d, l_1 = l + 1$.
Consider the case $d_1 = d - 1, l_1 = l$, that is, the degree estimate is off by 1 but the low degree l_1 is correct. In step 3.4 because we are using $\delta = 1$ additional evaluation point, we still have sufficient points to interpolate R/y^l. Thus

$$r = R/y^l \mod \Pi_{i=0}^N (y - \alpha_i) = c_d y^{d-l} + ... + c_l \bmod p.$$

Because p is not also unlucky (our assumption), then the algorithm will identify p_1 was unlucky with probability 1.

Now suppose $d_1 = d, l_1 = l + 1$, that is, the low degree estimate is off by 1. Again, in step 3.4, because we are using one additional evaluation point, we have sufficient points to interpolate R/y^l but the algorithm interpolates the rational function

$$r = R/y^{l+1} \mod \Pi_{i=0}^N (y - \alpha_i)$$

where $N = d - l$, $R/y^l = c_d y^{d-l} + ... + c_{l+1} y + c_l$ and

$$\Pi_{i=0}^N = y^{d-l} + ... + (-1)^{d-l} A$$

where $A = (-1)^{d-l}\alpha_0\alpha_1...\alpha_N$. Computing r over $\mathbb{Z}$ we find that $r = \frac{c_l}{A}y^d + ...$ Since α_i are chosen from $0 < \alpha_i < p$, i.e., the α_i are non-zero, A is also non-zero. Because p is not unlucky (our assumption), then p does not divide c_l, hence, the algorithm will identify that p_1 was unlucky with probability 1.

CASE $d_1 = d - 1, l_1 = l + 1$.
That is, the degree estimate and the low degree estimate are both off by 1. This time in step 3.4, we have one too few points to interpolate the rational function R/y^{l+1}, i.e., $N = d - l - 1$. Again, we interpolate

$$r = R/y^{l+1} \mod \Pi_{i=0}^{N}(y - \alpha_i)$$

where $R/y^l = c_d y^d + ... + c_{l+1}y + c_l$ and $\Pi_{i=0}^{N}(y - \alpha_i) = y^{d-l-1} - ... - A$ where $A = (-1)^{d-l}\alpha_0\alpha_1...\alpha_N$. Over $\mathbb{Z}$,

$$r = (c_d - \frac{c_l}{A})y^{d-l-1} + ...$$

Thus the algorithm fails to identify that p_1 is unlucky if and only if $Ac_d - c_l \equiv 0 \bmod p$. Now since the algorithm chooses α_i from $0 < \alpha_i < p$ at random A is non-zero and it is uniformly distributed on $(0, p)$. Since p is not unlucky (our assumption), $c_d \not\equiv 0 \bmod p$ and and $c_l \not\equiv 0 \bmod p$. Thus $\Pr(Ac_d - c_l \equiv 0 \bmod p) = 1/(p - 1)$.

CASE $d_1 = d - 2, l_1 = l$.
That is, the degree estimate is off by 2 and the low degree l_1 is correct. If $\delta = 1$ then in step 3.4 we have 1 too few points to interpolate R/y^l, i.e., $N = d - l - 1$. The value r interpolated in step 3.4 satisfies

$$r = R/y^l \mod \Pi_{i=0}^{N}(y - \alpha_i)$$

where $R/y^l = c_d y^{d-l} + c_{d-1}y^{d-l-1} + ... + c_l$ and

$$\Pi_{i=0}^{N} = y^{d-l-1} - Cy^{d-l-2} - ... - A$$

where $C = \alpha_0 + \alpha_1 + ... + \alpha_N$. Computing r over $\mathbb{Z}$ we have

$$r = (c_{d-1} - c_d C)y^{d-l-1} + ...$$

Because $N > 0$, i.e. we have at least two points, C is still almost uniformly distributed on $[0, p)$. For all $0 < x, y < p$ we have by symmetry

$$\Pr(C \equiv x \bmod p) = \Pr(C \equiv y \bmod p).$$

By computer experiment, we find that

$$\Pr(C = 0) - \Pr(C \neq 0) = \pm\frac{1}{(p-1)^N}.$$

Therefore C is almost uniformly distributed on $[0, p)$, thus $\Pr(c_{d-1} - c_d C) \equiv 0 \bmod p) \sim 1/p$. We conclude that if p is not also unlucky, the probability of not detecting this is $\sim 1/p$ in this case.

To summarize, if p_1 is unlucky, and $p_2, ..., p_K$ are not unlucky, then the probability that the algorithm fails to detect that p_1 is unlucky is decreasing from $(d-l)/(p-1)$ to $1/p$ as $\deg_y r_1$ increases. The maximum failure probability depends on the difference $d - l$. It is bounded by $(D - L)^K/P$ where P is the product of the K check primes. This maximum failure probability can be reduced by choosing $\delta > 1$. Then the maximum failure probability is bounded by $(D - L)^{\delta K}/P^{\delta}$.

4. A MONIC RESULTANT ALGORITHM

Let $R = c_d y^d + c_{d-1}y^{d-1} + ... + c_{l+1}y^{l+1} + c_l y^l$. Algorithm MRES below outputs the monic resultant R/c_d with high probability. It uses Wang's rational number reconstruction (see [13, 3, 10]) to recover the rational coefficients of R/c_d from R/c_d modulo M where M is a product of primes.

Algorithm MRES

Input $A, B \in \mathbb{Z}[x, y]\backslash\{0\}$ of degree m and n resp.
Input $D \geq L \geq 0$ satisfying $D \geq d$ and $L \leq l$.
Input $H \geq h = \max(1, |c_d|, |c_{d-1}|, ..., |c_{l+1}|, |c_l|)$.
Input $K \geq 1$ (number of primes of agreement.)
Output $R = \mathrm{res}_x(A, B) \in \mathbb{Z}[y]$.

1 Initialize $M = 1, \Delta = D - L, \delta = 0, l = L, d = D$.

2 Initialize S to a set of primes such that $\Pi_{p\in S} \gg H^2$, and for each $p \in S$, we have $p > \Delta + \deg_y A + \deg_y B$ and p is not a bad prime.

3 REPEAT

- 3.1 Choose a new prime p from S. Set $A_p = A \bmod p$ and $B_p = B \bmod p$.
- 3.2 Set $N = \Delta + \delta$ and choose $N+1$ distinct non-zero evaluation points $\alpha_0, \alpha_1, ..., \alpha_N$ from $\mathbb{Z}_p$ at random such that $a_m(\alpha_i) \not\equiv 0 \bmod p$ and $b_m(\alpha_i) \not\equiv 0 \bmod p$.
- 3.3 FOR $i = 0, 1, ..., N$ DO
  Compute $r_i \in \mathbb{Z}_p$ the resultant of $A_p(\alpha_i)$ and $B_p(\alpha_i)$ modulo p using the Euclidean algorithm and set $r_i = r_i/\alpha_i^l \bmod p$.
- 3.4 Interpolate $r \in \mathbb{Z}_p[y]$ from (α_i, r_i) for $i = 0..N$. Set $r = y^l r$.
- 3.4b If $r \neq 0$ then make r monic in $\mathbb{Z}_p[y]$.
- 3.5 IF $\delta = 0$ set the bounds:
  - 3.5.1 If $r = 0$ set $d = L - 1, l = L$ otherwise set $d = \deg_y(r)$ and l to the low-degree of r in y.
  - 3.5.1 Set $\delta = 1$, $\Delta = d - l$, $M = p$, $\bar{R} = r$, $G = FAIL$, $j = 0$.
- 3.6 ELIF $\deg_y(r) > d$ then – restart the algorithm
  Initialize $M = 1, \Delta = D - L, \delta = 0, l = L, d = D, G = FAIL$.
- 3.6b ELIF $\deg_y(r) < d$ then GOTO 3.1.
- 3.7 ELSE apply the Chinese remainder theorem:
  - 3.7.1 Solve $C \equiv r \bmod p$ and $C \equiv \bar{R} \bmod M$ for C.
  - 3.7.2 Set $M = p \times M, \bar{R} = C$.
- 3.8 IF $G \neq FAIL$ and $G \equiv r \bmod p$
  THEN set $j = j + 1$ ELSE set $j = 0, G = FAIL$.
- 3.9 IF $G = FAIL$ then apply rational reconstruction to the coefficients of $\bar{R}$ modulo M to obtain G.

UNTIL $j = K$.

4 Clear the fractions in G. Output G.

The first difference between algorithm MRES and PRES is that MRES makes the images monic in step 3.4b so that R/c_d modulo M is reconstructed in step 3.7. A second difference is the addition of step 3.6b. If the degree of an image is too low, this means the current prime p divides c_d and we cannot reconstruct the rational coefficients using this image. A third difference is that we do not put the image in the symmetric range in 3.7. The treatment of negative rational coefficients is handled by rational reconstruction.

The main difference is the use of rational number reconstruction in step 3.9. If this succeeds in step 3.9, the test in step 3.8 tests in the subsequent iterations whether the current image r is consistent with G the result of the rational reconstruction. The algorithm terminates when we have K consecutive primes of agreement.

The rational number reconstruction in step 3.9 should be done in such a way that it will fail with high probability when M is not large enough yet to recover the coefficients of R/c_d. Otherwise rational reconstruction may dominate the cost of the algorithm. How to do this is described in [10]. One may trivially modify the bounds used by Wang's rational reconstruction (for example, by using $\sqrt{(M-1)/8}$ instead of the default bounds $\sqrt{(M-1)/2}$ for the numerators and denominators) or use the rational reconstruction algorithm of Monagan in [10] to force it to fail with high probability when M is not large enough to reconstruct R/c_d.

Suppose the resultant $R = ay+b \in \mathbb{Z}[y]$ where $\gcd(a,b) = 1$ and a and b have the same length. To reconstruct the monic polynomial $y+b/a$ using rational reconstruction, algorithm MRES will need approximately twice as many primes as algorithm PRES. This is the reason for the H^2 in step 2. This suggests a hybrid algorithm, where we attempt to reconstruct both R and R/c_d, will be best.

The argument that algorithm MRES must always terminate is the same as was made for algorithm PRES. We argue that the failure probability of algorithm MRES is no worse than twice that of algorithm PRES. This is because the only essential difference is the use of rational reconstruction, and, as we have just remarked, this may require up to twice the number of primes.

5. CONCLUDING REMARKS

Examples 2.1 and 2.2 tell us that the modular resultant algorithm of Collins may perform poorly when the bounds for the coefficients and degrees are off. Example 2.4 gives a second reason, namely, if the resultant is sparse and in several variables. These realities make the choice between using the subresultant algorithm, a determinant based algorithm, or the modular resultant algorithm awkward. Algorithm PRES solves the first problem. For sparse resultants one might consider using a sparse interpolation algorithm. See [14, 8].

Examples 2.3 and 2.4 tell us that the height of the primitive part of the resultant can be much smaller than the height of the resultant even when the inputs are primitive. Algorithm MRES solves this problem.

An alternate design of algorithm PRES and algorithm MRES would be to incrementally interpolate the first image instead of using the bounds D and L. The main reason we chose not to do this is that incremental interpolation is significantly more expensive, especially if one does not know l. This can make the performance of the algorithm poor for the normal case when the degree bounds are good. Another practical reason was that on our test problems, the time spent computing the first image using D and L was not the main cost. Also, by knowing in advance how many points we are using we can more easily use an asymptotically fast interpolation algorithm when $D - L + 1$ is very large.

Acknowledgement

We acknowledge the contribution made by the following people who noticed various inefficiencies in Maple's implementation of Collins' algorithm in Maple: Chris Smith, Roman Pearce, and Greg Fee. We also wish to thank the two anonymous referees who contributed many useful remarks and suggestions for improving the paper.

6. REFERENCES

[1] W. S. Brown, J. F. Traub. On Euclid's Algorithm and the Theory of Subresultants. *JACM* **18**(4), 505–514, 1971.

[2] George Collins. The Calculation of Multivariate Polynomial Resultants. *JACM* **18**(4), 515–532, 1971.

[3] G. E. Collins and M. J. Encarnacion. Efficient Rational Number Reconstruction. *J. Symbolic Comp.* **20**, pp. 287–297, 1995.

[4] Karin Gatermann, "Polynomial Systems with Symmetry", talk presented at "Solving Systems of Equations", September 14-18, 1998, MSRI.

[5] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, UK, 1999.

[6] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publ., Boston, Massachusetts, USA, 1992.

[7] A. Goldstein, G. Graham. A Hadamard type bound on the coefficients of a determinant of polynomials. *SIAM Review*, **16**, pp. 394–395, 1974.

[8] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comp.* **9**, pp. 301–320, 1990.

[9] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, P. DeMarco. *Maple 9 Introductory Programming Guide* Waterloo Maple, 2003. ISBN: 1-894511-43-3.

[10] M. B. Monagan. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '2004*, ACM Press, pp. 243–249, 2004.

[11] Posso test suite: http://www-sop.inria.fr/saga/POL/

[12] B. Trager. Algebraic Factoring and Rational Function Integration. *Proceedings of SYMSAC '76*, ACM Press, pp. 219–226, 1976.

[13] P. S. Wang, M. J. T. Guy, J. H. Davenport. p-adic Reconstruction of Rational Numbers. SIGSAM Bulletin, **16**(2), 1982.

[14] R. Zippel. Interpolating Polynomials from their Values. *J. Symbolic Comp.* **9**(3), 375–403, 1990.