

University of Texas Rio Grande Valley

ScholarWorks @ UTRGV

---

Theses and Dissertations - UTB/UTPA

---

12-2005

## An improvement and a generalization of Zippel's sparse multivariate polynomial interpolation algorithm

Michael D. Brazier

*University of Texas-Pan American*

Follow this and additional works at: [https://scholarworks.utrgv.edu/leg\\_etd](https://scholarworks.utrgv.edu/leg_etd)



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Brazier, Michael D., "An improvement and a generalization of Zippel's sparse multivariate polynomial interpolation algorithm" (2005). *Theses and Dissertations - UTB/UTPA*. 781.  
[https://scholarworks.utrgv.edu/leg\\_etd/781](https://scholarworks.utrgv.edu/leg_etd/781)

This Thesis is brought to you for free and open access by ScholarWorks @ UTRGV. It has been accepted for inclusion in Theses and Dissertations - UTB/UTPA by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact [justin.white@utrgv.edu](mailto:justin.white@utrgv.edu), [william.flores01@utrgv.edu](mailto:william.flores01@utrgv.edu).

AN IMPROVEMENT AND A GENERALIZATION OF  
ZIPPEL'S SPARSE MULTIVARIATE POLYNOMIAL  
INTERPOLATION ALGORITHM

A Thesis by  
MICHAEL D. BRAZIER

Submitted to the Graduate School  
of the University of Texas Pan American  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE


December 2005

Advisor: Dr. Arthur Chtcherba  
Major Subject: Computer Science  
*2000 Mathematics Subject Classification*  
Primary 68W40 Secondary 12Y05, 65D05

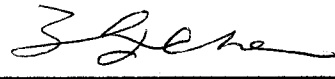
AN IMPROVEMENT AND A GENERALIZATION OF  
ZIPPEL'S SPARSE MULTIVARIATE POLYNOMIAL  
INTERPOLATION ALGORITHM

A Thesis by  
MICHAEL D. BRAZIER

Approved as to style and content by:



Dr. Arthur Chtcherba  
Chair of Committee



Dr. Zhixiang Chen  
Committee Member



Dr. Roger Knobel  
Committee Member

December 2005

## Abstract

Brazier, Michael D. An Improvement and a Generalization of Zippel's Sparse Multivariate Polynomial Interpolation Algorithm. Master of Science (MS), December 2005. 39 pp., 2 tables, 3 algorithms, references, 16 titles.

The algorithm most often used for the problem of interpolating sparse multivariate polynomials from their values is Zippel's probabilistic algorithm (1988). The algorithm evaluates the function to be interpolated at a significant number of points, and for many problems of interest processing evaluations dominates the running time. This thesis presents an improvement of Zippel's algorithm, which decreases the number of evaluations needed for an interpolation by using transposed Vandermonde systems for the univariate interpolation step of Zippel's algorithm. The technique also allows a more general form of the algorithm: it becomes possible to interpolate more than one variable within a single stage of Zippel's method.

## Acknowledgements

I wish to express my appreciation and gratitude to:

- Dr. Arthur Chitcheba, my advisor, for making my graduate study possible, and for his academic guidance;
- The Computing and Information Technology Center at the University of Texas-Pan American, Dr. Deepak Kapur of the University of New Mexico, and the National Science Foundation<sup>1</sup> for their support and assistance;
- My parents, Gerald and Pearl Brazier.

---

<sup>1</sup>Grant #CCR-0203051

## Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Algorithms	vi
List of Tables	vi
1 Introduction	1
2 Zippel's Algorithm	4
2.1 Description . . . . .	4
2.2 Example . . . . .	8
2.3 Analysis . . . . .	10
3 Univariate Interpolation with Vandermonde Matrices	13
3.1 Empirical Data . . . . .	16
4 Using Interpolant Images in Several Variables	19
4.1 The Algorithm . . . . .	20
4.2 Example . . . . .	22
4.3 Analysis . . . . .	25
5 Conclusions	27
6 Related Work	28
Bibliography	29
Vita	33

## List of Algorithms

1	Zippel's Algorithm . . . . .	5
2	Zippel's Algorithm, with Vandermonde univariate interpolation	13
3	Zippel's Algorithm, generalized for multivariate images . . . .	18

## List of Tables

1	Sample Problems . . . . .	16
2	Comparing Univariate Interpolation Methods . . . . .	19

## 1 Introduction

Problems from many areas, such as geometric reasoning, computer modelling, implicitization, computer vision, robotics and kinematics [Man92, Kap96, Cht03] can be expressed as a system of polynomials, for which the desired solutions are values of the variables at which all the polynomials vanish at once. In practice the most efficient ways to solve a system of polynomials involve finding its resultant: a single polynomial which vanishes if and only if the system has a non-trivial solution. Several methods for constructing resultants have been described in the last century, by Macaulay [Mac02, Mac16], Sylvester [Syl53] and Dixon [Dix08, KSY94]. In all of these the resultant is expressed as the determinant of a matrix of polynomials, and this determinant must be expanded and factored to discover the solution to the original system.

A direct computation of the determinant of a matrix of polynomials, us-



ing Gaussian elimination on the entries, proves to be an algorithm of exponential complexity, due to the phenomenon called "intermediate expression swell" – meaning that the polynomials calculated during the Gaussian elimination have much larger coefficients, many more terms, or both, than the polynomials in the input, or the final result.

It is more efficient to use an algorithm for multivariate interpolation. The matrix becomes a "black box", which is evaluated by substituting values for each variable and returning the determinant of the resulting numerical matrix (this requires only polynomial time –  $O(n^3 * p(n))$  for an  $n * n$  matrix in which  $p(n)$  is the most complex entry.) Then the polynomial structure of the symbolic determinant is recovered by evaluating the black box at appropriate tuples of values and interpolating the results. For a polynomial in  $n$  variables  $x_1 \dots x_n$ , in which the variables are of degrees  $d_1 \dots d_n$ , an algorithm might conceivably have to perform  $\prod_{j=1}^n (d_j + 1)$  evaluations, but the polynomial systems which arise in practice almost never have such dense resultants. Resultant polynomials are normally sparse: the number of terms in them is far less than the theoretical upper bound.

The first algorithm for multivariate interpolation that takes advantage of sparsity in the interpolant was discovered by Richard Zippel [Zip93] in 1979. It is a Monte Carlo algorithm (correct with high probability) and interpolates variable by variable (using univariate interpolation) so it is also

sensitive to the number of variables in the interpolant. (Section 2 describes this algorithm in detail.)

In 1988 Michael Ben-Or and Prasoona Tiwari found a different algorithm for multivariate interpolation [BOT88] based on the Berlekamp-Massey algorithm for finding the generating polynomial of a linear recurrence. Ben-Or and Tiwari's algorithm, unlike Zippel's, is not randomized and interpolates in a single pass, regardless of the number of variables in the interpolant. However, it requires an upper bound on the number of terms in the interpolant. When the interpolant is a determinant, the best upper bound available proves to be on the order of the product of the degrees of all the variables – a bound no better than the worst case for Zippel's algorithm. In addition, Ben-Or and Tiwari's algorithm calculates numbers of the size of an evaluated term in the interpolant, which leads to an extreme intermediate expression swell if one uses infinite-precision arithmetic, and forces a very large modulus if one works instead in a finite field. In Zippel's algorithm the largest numbers needed are on the order of the size of the coefficients, so one can work in finite fields with somewhat smaller moduli.

Both Zippel's and Ben-Or and Tiwari's algorithms have been improved and generalized in several ways. Erich Kaltofen and Lakshman Yagati, by finding quasi-linear algorithms to solve Toeplitz and transposed Vandermonde systems of equations [KL88], improved the time complexities of

both interpolation algorithms. In a later paper [KLL00] Kaltofen, Wen-shin Lee and Austin A. Lobo presented a randomized version of Ben-Or and Tiwari's algorithm which does not require an upper bound on the number of terms, then used this algorithm for the univariate interpolations in Zippel's algorithm.

This thesis offers another improvement and generalization to Zippel's algorithm. In Section 3 we give an improvement to Zippel's algorithm which, by further exploiting sparsity in the interpolant, decreases the number of evaluations needed for interpolation; and give empirical data showing a speed-up on test problems. In Section 4 we introduce a generalization of Zippel's algorithm, suggested by the technique in Section 3.

## 2 Zippel's Algorithm

### 2.1 Description

Algorithm 1 is Zippel's original algorithm. The first step (at line 10) is to fix, for each variable  $x_j$  in the interpolant, a random element  $b_j$  of the base field; a prime  $p_j$ , and  $d_j$  an upper bound on the degree of  $x_j$  in the interpolant. This last (named  $\text{degree}(I, x_j)$  in Algorithm 1) may be determined by inspection of the interpolant's structure, or (if this isn't possible) by an adaptive univariate interpolation, such as the "early termination" algorithms described by Kaltofen and Lee [KL03].

---

**Algorithm 1:** Zippel's Algorithm

---

**Data:**  $\mathcal{Q}$  a field  $I(x_1 \dots x_n)$  (the *interpolant*) an expression in the variables  $x_1 \dots x_n$

**Result:**  $\forall a_1 \dots a_n \in \mathcal{Q}, P(a_1 \dots a_n) \in \mathcal{Q}[x_1 \dots x_n] = I(a_1 \dots a_n)$

```

1  Vandermonde( $P \in \mathcal{Q}[\bar{x} \subset \{x_1 \dots x_n\}], \bar{a} \in \mathcal{Q}^{|\bar{x}|}$ ): begin
2     $t \leftarrow \text{nTerms}(P)$ ;
3     $M \leftarrow \text{Vector}(\mathcal{Q}[\bar{x}], 1 \dots t)$ ;
4     $V \leftarrow \text{Matrix}(\mathcal{Q}, 1 \dots t, 1 \dots t)$ ;
5    for  $j = 1 \dots t$  do
6       $M_j \leftarrow \text{monomial}(j, P)$ ;
7      for  $k = 1 \dots t$  do  $V_{kj} \leftarrow M_j(\bar{a})^k$ ;
8    return  $M, V$ ;
9  end

10 for  $j = 1 \dots n$  do
11    $d_j \leftarrow \text{degree}(I, x_j)$ ;
12    $p_j \leftarrow$  a prime  $< |\mathcal{Q}|$ ,  $\neq p_i, i < j$ ;
13    $b_j \leftarrow \text{random}(\mathcal{Q})$ ;
14  $P_0 \leftarrow I(b_1 \dots b_n)$ ;
15 for  $j = 0 \dots n - 1$  do
16    $(M_j, V) \leftarrow \text{Vandermonde}(P_j, \{p_1 \dots p_j\})$ ;
17    $E \leftarrow \text{Matrix}(1 \dots |M_j|, 0 \dots d_{j+1})$ ;
18   for  $i = 0 \dots d_{j+1}$  do
19      $y_i \leftarrow \text{random}(\mathcal{Q})$ ;
20     for  $t = 1 \dots |M_j|$  do
21        $E_{ti} \leftarrow I(p_1^i \dots p_j^i, y_i, b_{j+2} \dots b_n)$ ;
22    $Z \leftarrow V^{-1} \times E$ ;
23    $F \leftarrow \text{Vector}(\mathcal{Q}[x_{j+1}], 1 \dots |M_j|)$ ;
24   for  $t = 1 \dots |M_j|$  do
25      $F_t(x_{j+1}) \leftarrow \text{UnivarInterp}((y_0, Z_{t,0}) \dots (y_{d_{j+1}}, Z_{t,d_{j+1}}))$ ;
26    $P_{j+1} \leftarrow M \cdot F$ ;
27  $P \leftarrow P_n$ ;

```

---

Once  $\{d_1 \dots d_n\}$ ,  $\{p_1 \dots p_n\}$ ,  $\{b_1 \dots b_n\}$ , and an initial polynomial  $P_0$  are fixed, the algorithm begins its main loop at line 15. Each pass of this loop takes an *image*  $P_k$  of the interpolant  $I$  in the first  $k$  of the  $n$  variables – by which is meant, that

$$\forall a_1 \dots a_k \in \mathcal{Q}, P_k(a_1 \dots a_k) = I(a_1 \dots a_k, b_{k+1} \dots b_n).$$

(So  $P_0 = I(b_1 \dots b_n)$ , a constant polynomial, and  $P_n$  is the target polynomial.) This image is extended by the next variable  $x_{k+1}$  to produce the next image  $P_{k+1}$ . Repeating the loop body  $n$  times, starting from the constant  $P_0$ , calculates  $P_n = P$ , the target.

The image  $P_k$  may be written as a sum of terms:

$$P_k = \sum_{j=1}^J c_j M_{jk}(x_1 \dots x_k)$$

where each  $M_{jk}$  is a monomial, a product of powers of the variables. The image  $P_{k+1}$  may also be written in terms of the  $M_{jk}$  monomials:

$$P_{k+1} = \sum_{j=1}^J f_j(x_{k+1}) M_{jk}(x_1 \dots x_k),$$

and to extend  $P_k$  to  $P_{k+1}$  we must find the polynomials  $f_j$ . The method is to substitute for the  $x_1 \dots x_k$  powers of the distinct primes  $p_1 \dots p_k$  in the monomials  $M_{j,k}$ , and in  $I$ , which gives a system of linear equations:

$$\begin{aligned} \sum_{j=1}^J f_j(x_{k+1}) M_{j,k}(p_1^0 \dots p_k^0) &= I(p_1^0 \dots p_k^0, x_{k+1}, b_{k+2} \dots b_n) \\ \sum_{j=1}^J f_j(x_{k+1}) M_{j,k}(p_1^1 \dots p_k^1) &= I(p_1^1 \dots p_k^1, x_{k+1}, b_{k+2} \dots b_n) \\ \sum_{j=1}^J f_j(x_{k+1}) M_{j,k}(p_1^2 \dots p_k^2) &= I(p_1^2 \dots p_k^2, x_{k+1}, b_{k+2} \dots b_n) \\ &\vdots \\ \sum_{j=1}^J f_j(x_{k+1}) M_{j,k}(p_1^J \dots p_k^J) &= I(p_1^J \dots p_k^J, x_{k+1}, b_{k+2} \dots b_n) \end{aligned}$$

To recover the coefficients of each  $f_j$ , we solve this system for  $d_{k+1} + 1$  values  $x_{k+1,0} \dots x_{k+1,d_{k+1}}$ . This is equivalent to solving the matrix equation  $V \times Z = E$ , where

$$V = \begin{bmatrix} M_{1,k}(p_1 \dots p_k)^0 & \dots & M_{J,k}(p_1 \dots p_k)^0 \\ M_{1,k}(p_1 \dots p_k)^1 & \dots & M_{J,k}(p_1 \dots p_k)^1 \\ \vdots & \ddots & \vdots \\ M_{1,k}(p_1 \dots p_k)^J & \dots & M_{J,k}(p_1 \dots p_k)^J \end{bmatrix}$$

is a transposed Vandermonde matrix,

$$Z = \begin{bmatrix} f_1(x_{k+1,0}) & \dots & f_1(x_{k+1,d_{k+1}}) \\ f_2(x_{k+1,0}) & \dots & f_2(x_{k+1,d_{k+1}}) \\ \vdots & \ddots & \vdots \\ f_J(x_{k+1,0}) & \dots & f_J(x_{k+1,d_{k+1}}) \end{bmatrix}$$

holds the values of  $f_1 \dots f_J$  at  $x_{k+1,0} \dots x_{k+1,d_{k+1}}$ , and

$$E = \begin{bmatrix} I(p_1^0 \dots p_k^0, x_{k+1,0}, b_{k+2} \dots b_n) & \dots & I(p_1^0 \dots p_k^0, x_{k+1,d_{k+1}}, b_{k+2} \dots b_n) \\ I(p_1^1 \dots p_k^1, x_{k+1,0}, b_{k+2} \dots b_n) & \dots & I(p_1^1 \dots p_k^1, x_{k+1,d_{k+1}}, b_{k+2} \dots b_n) \\ \vdots & \ddots & \vdots \\ I(p_1^J \dots p_k^J, x_{k+1,0}, b_{k+2} \dots b_n) & \dots & I(p_1^J \dots p_k^J, x_{k+1,d_{k+1}}, b_{k+2} \dots b_n) \end{bmatrix}$$

holds the corresponding values of  $I$ . Because  $V$  is a transposed Vandermonde matrix, solving for  $Z$  does not require general matrix multiplication, but can be done in  $O(d_{k+1}J(\log J)^2)$  time, or  $O(d_{k+1}J^2)$  time, whichever is faster (see [PFTV90] for the  $O(n^2)$  algorithm, and [KL88] for the  $O(n(\log n)^2)$  algorithm.)

To begin the loop body, the algorithm calls the function **Vandermonde** (defined at line 1) which extracts from  $P_k$  a list of its monomials  $M_{jk}$  and constructs the matrix  $V$  by substituting the primes  $p_1 \dots p_k$  into them. Next it fills the matrix  $E$  with evaluations of  $I$ ; then (at line 22) it solves  $V \times Z = E$

for  $Z$ . Finally (at line 24)  $J$  univariate interpolations, taking the rows of  $Z$  for the polynomial values, recover the polynomials  $f_1(x_{k+1}) \dots f_J(x_{k+1})$ ; and  $\sum_{j=1}^J f_j(x_{k+1}) * M_{j,k} = P_{k+1}$ .

## 2.2 Example

Suppose that  $I(w, x, y, z) = w^2x^3yz - 3wx^3yz^2 - w^2xy^2z + wx^3z^2 - 5xy^2z^2 + 2x^3z^2 + 2wx^3 - wx + 2z^2 - 4w$ . The degrees of  $I$  in  $(w, x, y, z)$  are, respectively,  $(2, 3, 2, 2)$ ; we pick the primes  $(2, 3, 5, 7)$  and anchors  $(11, 13, 17, 19)$ , and set  $P_0 = 1$ , which brings us to line 15 of Algorithm 1 for the first stage. We may interpolate the variables in any order; for this example, we'll use the order  $x, w, y, z$ .

### Stage 1: $x$

$\{M_{i0}\} = \{1\}$  and  $\text{degree}(I, w) = 3$ , so  $V = 1$ ,

$$E = [I(11, x_0, 17, 19), I(11, x_1, 17, 19), I(11, x_2, 17, 19), I(11, x_3, 17, 19)],$$

and the loop at line 24 performs 1 interpolation, yielding

$$P_1 = -158723x^3 - 1186067x + 678.$$

### Stage 2: $w$

$\{M_{i1}\} = \{1, x, x^3\}$  and  $\text{degree}(P, w) = 2$ , so

$$V = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3^1 & 3^3 \\ 1^2 & (3^1)^2 & (3^3)^2 \end{bmatrix},$$

$$E = \begin{bmatrix} I(w_0, 3^0, 17, 19) & I(w_1, 3^0, 17, 19) & I(w_2, 3^0, 17, 19) \\ I(w_0, 3^1, 17, 19) & I(w_1, 3^1, 17, 19) & I(w_2, 3^1, 17, 19) \\ I(w_0, 3^2, 17, 19) & I(w_1, 3^2, 17, 19) & I(w_2, 3^2, 17, 19) \end{bmatrix},$$

and the loop at line 24 performs 3 interpolations, yielding

$$P_2 = (323w^2 - 18048w + 722)x^3 - (5491w^2 - w - 521645)x - 4w + 722.$$

**Stage 3: y**

$\{M_{i2}\} = \{1, w, x, wx, w^2x, x^3, wx^3, w^2x^3\}$  and  $\text{degree}(P, y) = 2$ , so

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2^1 & 3^1 & 2^1 3^1 & 2^2 3^1 & 3^3 & 2^1 3^3 & 2^2 3^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1^7 & (2^1)^7 & (3^1)^7 & (2^1 3^1)^7 & (2^2 3^1)^7 & (3^3)^7 & (2^1 3^3)^7 & (2^2 3^3)^7 \end{bmatrix},$$

$$E = \begin{bmatrix} I(2^0, 3^0, y_0, 19) & I(2^0, 3^0, y_1, 19) & I(2^0, 3^0, y_2, 19) \\ I(2^1, 3^1, y_0, 19) & I(2^1, 3^1, y_1, 19) & I(2^1, 3^1, y_2, 19) \\ \vdots & \vdots & \vdots \\ I(2^7, 3^7, y_0, 19) & I(2^7, 3^7, y_1, 19) & I(2^7, 3^7, y_2, 19) \end{bmatrix},$$

and the loop at line 24 performs 8 interpolations, yielding

$$P_3 = (19y)w^2x^3 + (-1083y + 363)wx^3 + 722x^3 - (19y^2)w^2x - wx - (1805y^2)x - 4w + 722.$$

**Stage 4: z**

$\{M_{i3}\} = \{1, w, wx, xy^2, x^3, wx^3, wx^3y, w^2xy^2, w^2x^3y\}$  and  $\text{degree}(P, z) =$

2, so

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2^1 & 2^1 3^1 & 3^1 5^1 & 3^3 & 2^1 3^3 & 2^1 3^3 5^1 & 2^2 3^1 5^2 & 2^2 3^3 5^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1^8 & (2^1)^8 & (2^1 3^1)^8 & (3^1 5^2)^8 & (3^3)^8 & (2^1 3^3)^8 & (2^1 3^3 5^1)^8 & (2^2 3^1 5^2)^8 & (2^2 3^3 5^1)^8 \end{bmatrix},$$

$$E = \begin{bmatrix} I(2^0, 3^0, 5^0, z_0) & I(2^0, 3^0, 5^0, z_1) & I(2^0, 3^0, 5^0, z_2) \\ I(2^1, 3^1, 5^1, z_0) & I(2^1, 3^1, 5^1, z_1) & I(2^1, 3^1, 5^1, z_2) \\ \vdots & \vdots & \vdots \\ I(2^8, 3^8, 5^8, z_0) & I(2^8, 3^8, 5^8, z_1) & I(2^8, 3^8, 5^8, z_2) \end{bmatrix},$$



and the loop at line 24 performs 9 interpolations, yielding

$$P_3 = (z)w^2x^3y - (z)w^2xy^2 - (3z^2)wx^3y + (z^2+2)wx^3 + (2z^2)x^3 - (5z^2)xy^2 - wx - 4w + 2z^2 = I.$$

### 2.3 Analysis

For an interpolant  $I$  of  $n$  variables, Zippel's algorithm runs the main loop at line 15  $n$  times. Within that loop it evaluates  $I|M_j|(d_{j+1} + 1)$  times to calculate  $E$ . It then calculates  $V^{-1} \times E$ , which involves  $d_{j+1} + 1$  solutions of a  $|M_j| \times |M_j|$  transposed Vandermonde system. Finally it calls `UnivarInterp`  $|M_j|$  times, calculating polynomials of degree  $d_{j+1}$ .  $|M_j| \leq t$ , the number of terms in the target polynomial, and we may define  $d \equiv \max_{j=1}^n d_j$ ; then one call of `UnivarInterp` is  $O(d^2)$ , one Vandermonde solution is  $O(t \log^2 t)$ , and the complexity of Zippel's algorithm is

$$O(ndt(O(I) + \log^2 t + d))$$

where  $O(I)$  is the complexity of evaluating  $I$  once.

Zippel's algorithm assumes that if, in an image polynomial

$$P_k = \sum_{j=1}^J c_j M_{jk}(x_1 \dots x_k)$$

one of the coefficients  $c_j$  is zero, then in the target polynomial, which may be written

$$P = \sum_{j=1}^J f_j(x_{k+1} \dots x_n) M_{jk}(x_1 \dots x_k)$$

the corresponding  $f_j$  is the zero polynomial. (For example, if the monomial  $x_1^2 x_2^3$  in  $P_2$  has 0 for a coefficient, the monomials starting with  $x_1^2 x_2^3 \dots$  would be expected to vanish in  $P$ .) Therefore, when  $P_k$  is extended to  $P_{k+1}$ , each coefficient appearing in the  $f_j$ 's which proves to be 0 indicates a monomial to be omitted from  $P_{k+1}$ , decreasing the work needed in later stages.

If it so happens that  $f_j(b_{k+1} \dots b_n) = 0$  – that is, if  $f_j$  is a non-zero polynomial that evaluates to 0 at the anchor – then Zippel's algorithm, as it calculates  $P_{k+1}$ , will incorrectly omit those terms which are images of  $f_j M_{j,k}$  in  $P$ . Zippel proved [Zip90] an upper bound on the probability that a randomly chosen anchor would be a zero for one of a set of polynomials:

**Theorem 2.1 (Zippel)** *Let  $f_1, f_2 \dots f_s$  be elements of  $k[X_1, X_2 \dots X_n]$ , where the degree in each variable is bounded by  $d$ . Let  $\mathcal{P}(f_1, f_2 \dots f_s)$  be the probability that a randomly chosen point  $\bar{x}$  is a zero of any of the  $f_i$ , where  $x_i$  is an element of a set with  $q$  elements. Then*

$$\mathcal{P}(f_1, f_2 \dots f_s) < \frac{nds}{q}.$$

At each stage of Zippel's algorithm there are no more than  $t$  polynomials, in less than  $n$  variables, each of degree less than  $d$ , which must not be 0 at the anchor for the stage to compute the correct image. Since there are  $n$  stages, by the theorem 2.1 the probability of failure is

$$O\left(\frac{n^2 dt}{q}\right).$$

The other possible error in Zippel's algorithm arises when calculating in finite fields. The matrix equation  $V \times Z = E$  (solved at line 22) has no solution if  $V$  is a singular matrix, which occurs if two distinct monomials  $M_{i,k}, M_{j,k}$  in  $P_k$  evaluate to the same number. In a field of characteristic 0 we can just choose distinct primes for each variable to guarantee that different monomials evaluate to different numbers. In a finite field  $\mathcal{Q}$ , however, an  $M_{j,k}$  may evaluate (in  $\mathbf{Z}$ ) to a number larger than the field's characteristic, which would mean (in  $\mathcal{Q}$ ) a collision with other monomials. In the same paper Zippel proved an upper bound on the probability of this event:

**Theorem 2.2 (Zippel)** *Let  $\bar{e}_1 \dots \bar{e}_T$  be  $n$ -tuples where each component is less than  $d$ . There exist no more than*

$$\frac{dT(T-1)(|\mathcal{Q}|-1)^{(n-1)}}{2}$$

*$n$ -tuples  $\bar{x}$  with components in  $\mathcal{Q}$  such that for some  $i$  and  $j$   $\bar{x}^{\bar{e}_i}$  and  $\bar{x}^{\bar{e}_j}$  have the same values. The probability that a randomly chosen  $\bar{x}$  will cause two of the  $\bar{x}^{\bar{e}_i}$  to have the same value is*

$$\frac{dT(T-1)}{2(|\mathcal{Q}|-1)}.$$

Each stage of Zippel's algorithm builds 1 Vandermonde matrix, so it follows that:

**Theorem 2.3 (Zippel)** *Let  $P$  be a polynomial in  $n$  variables, each of degree no more than  $d$  and with  $t(\gg n)$  non-zero terms. Assume the coefficients*

of  $P$  lie in a finite field with  $q$  elements. The probability that the sparse interpolation algorithm will give the wrong answer for this polynomial is less than

$$\frac{ndt^2}{q}.$$

### 3 Univariate Interpolation with Vandermonde Matrices

---

**Algorithm 2:** Zippel's Algorithm, with Vandermonde univariate interpolation

---

**Data:**  $\mathcal{Q}$  a field,  $I(x_1 \dots x_n)$  an expression in the variables  $x_1 \dots x_n$   
**Result:**  $\forall a_1 \dots a_n \in \mathcal{Q}, P(a_1 \dots a_n) \in \mathcal{Q}[x_1 \dots x_n] = I(a_1 \dots a_n)$

```

1 for  $j = 1 \dots n$  do
2    $d_j \leftarrow \text{degree}(I, x_j)$ ;
3    $p_j \leftarrow$  a prime  $< |\mathcal{Q}|, \neq p_i, i < j$ ;
4    $b_j \leftarrow \text{random}(\mathcal{Q})$ ;
5 for  $j = 1 \dots n$  do
6   for  $i = 0 \dots d_j$  do
7      $y_i \leftarrow \text{random}(\mathcal{Q})$ ;
8      $f_i \leftarrow I(b_1 \dots b_{j-1}, y_i, b_{j+1} \dots b_n)$ ;
9    $P_{x_j} \leftarrow \text{UnivarInterp}((y_0, f_0) \dots (y_{d_j}, f_{d_j}))$ ;
10  $P_1 \leftarrow P_{x_1}$ ;
11 for  $j = 2 \dots n$  do
12    $(M_{j-1}, V) \leftarrow \text{Vandermonde}(P_{j-1}, \{p_1 \dots p_{j-1}\})$ ;
13    $(M_{x_j}, W) \leftarrow \text{Vandermonde}(P_{x_j}, \{p_j\})$ ;
14    $E \leftarrow \text{Matrix}(1 \dots |M_{j-1}|, 1 \dots |M_{x_j}|)$ ;
15   for  $t = 1 \dots |M_{j-1}|$  do
16     for  $d = 1 \dots |M_{x_j}|$  do
17        $E_{td} \leftarrow I(p_1^d \dots p_{j-1}^d, y_d^t, b_{j+1} \dots b_n)$ ;
18    $C \leftarrow V^{-1} \times E \times (W^{-1})^\top$ ;
19    $P_j \leftarrow M_{j-1}^\top \times C \times M_{x_j}$ ;
20  $P \leftarrow P_n$ ;
```

---

It is possible to interpolate a univariate polynomial by solving a transposed Vandermonde system. If  $f(x) = \sum_{j=0}^D c_j x^j$ , then substituting powers of a random integer  $p$  for  $x$  gives a transposed Vandermonde system in the  $c_j$ 's:

$$\begin{aligned} \sum_{j=0}^D c_j p^{0 \cdot j} &= f(p^0), \\ \sum_{j=0}^D c_j p^{1 \cdot j} &= f(p^1), \\ &\vdots \\ \sum_{j=0}^D c_j p^{D \cdot j} &= f(p^D), \end{aligned}$$

or, in matrix form:

$$\begin{bmatrix} p^0 & \dots & p^{0 \cdot D} \\ p^1 & \dots & p^{1 \cdot D} \\ \vdots & \ddots & \vdots \\ p^D & \dots & p^{D \cdot D} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_D \end{bmatrix} = \begin{bmatrix} f(p^0) \\ f(p^1) \\ \vdots \\ f(p^D) \end{bmatrix}$$

Why would we want to use this method, instead of Newton's? If a coefficient  $c_k$  of  $f(x)$  is known beforehand to equal zero, then it can be omitted from every equation in the system, and one equation is redundant and can be dropped. The result remains a transposed Vandermonde system:

$$\begin{bmatrix} p^0 & \dots & p^{0 \cdot (k-1)} & p^{0 \cdot (k+1)} & \dots & p^{0 \cdot D} \\ p^1 & \dots & p^{1 \cdot (k-1)} & p^{1 \cdot (k+1)} & \dots & p^{1 \cdot D} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ p^{D-1} & \dots & p^{(D-1) \cdot (k-1)} & p^{(D-1) \cdot (k+1)} & \dots & p^{(D-1) \cdot D} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{D-1} \end{bmatrix} = \begin{bmatrix} f(p^0) \\ f(p^1) \\ \vdots \\ f(p^{D-1}) \end{bmatrix}$$

In the general case, of course, we do not know that any coefficients are zero before solving the system. But in stage  $k$  of Zippel's algorithm, if  $P_k$  has  $t_k$  non-zero terms,  $t_k$  univariate interpolations must be performed. If the image of  $P$  in the current variable  $x_{k+1}$  alone has a zero coefficient for the term  $x_{k+1}^j$ , then with high probability every monomial in  $P$  of degree  $j$  in

$x_{k+1}$  has a zero coefficient. To prove this, collect the terms of  $P$  into powers of  $x_{k+1}$ :

$$P = \sum_{j=0}^{d_{k+1}} M_j(x_1 \dots x_k, x_{k+2} \dots x_n) x_{k+1}^j.$$

There are  $d_{k+1}$  polynomials  $M_j$  in  $n-1$  variables in this sum, in which each variable's degree is no greater than  $d \geq d_{k+1}$ . Then the probability that any non-zero  $M_j$  evaluates to 0 at the anchor, by Theorem 2.1, is less than  $\frac{(n-1)dd_{k+1}}{q} \approx \frac{nd^2}{q}$ .

It follows that in each univariate interpolation in stage  $k$ , the coefficient for  $x_{k+1}^j$  may be omitted, and thus  $t_k$  interpolant evaluations can be saved, for each coefficient equal to 0 in  $P$ 's image in  $x_{k+1}$  – if we interpolate by solving transposed Vandermonde systems.

Interpolating  $t_k$  univariate polynomials  $f_1 \dots f_{t_k}$  with the same set of exponents  $\bar{e}_i (i \in \{1 \dots I\})$  is, then, equivalent to solving the matrix equation  $W \times C = Z$ , where

$$W = \begin{bmatrix} (p^{e_1})^0 & \dots & (p^{e_I})^0 \\ (p^{e_1})^1 & \dots & (p^{e_I})^1 \\ \vdots & \ddots & \vdots \\ (p^{e_1})^I & \dots & (p^{e_I})^I \end{bmatrix}$$

is a transposed Vandermonde matrix,

$$C = \begin{bmatrix} c_{0,1} & \dots & c_{I,1} \\ c_{0,2} & \dots & c_{I,2} \\ \vdots & \ddots & \vdots \\ c_{0,t_k} & \dots & c_{I,t_k} \end{bmatrix}$$

Table 1: Sample Problems

#	Problem Name	# Vars	Matrix Size	Rank	# Terms	Degree
1	CondPerpendXConicCircle	9	15×15	15	3614	16
2	ConfAnalCyclicMoleculeGH	16	28×25	23	55998	61
3	DistXConicO	13	5×5	5	6548	14
4	Emirs94exp	12	36×30	28	734	28
5	KissingCirclesProblem-dix	4	52×47	32	2176	46
6	LSY89-dix	8	12×12	12	1460	32
7	PappusTheoremProblem-dix	9	5×5	5	183796	32
8	PDE2subs2-dixdia	2	359×361	350	292250	2656
9	SumSquares5	6	462×462	462	20349	32
10	TetrahedronVol-dix	5	16×16	14	434	12

holds the coefficients of  $f_1 \dots f_{t_k}$ , and

$$Z = \begin{bmatrix} f_1(x_1) & \dots & f_1(x_I) \\ f_2(x_1) & \dots & f_2(x_I) \\ \vdots & \ddots & \vdots \\ f_{t_k}(x_1) & \dots & f_{t_k}(x_I) \end{bmatrix}$$

holds the values of  $f_1 \dots f_{t_k}$  at  $I$  arbitrary values  $x_1 \dots x_I$ .

Algorithm 2 makes these changes from Algorithm 1: at line 5 a new loop computes the images  $P_{x_1} \dots P_{x_n}$  of  $I$  in each variable  $x_1 \dots x_n$ . The stage loop starts (line 11) with  $P_1 = P_{x_1}$ , not  $P_0$ . And at line 15, instead of solving  $V \times E = Z$ , we solve  $V \times Z \times W^\top = E$ , where  $W$  is the matrix for interpolating the variable  $x_{k+1}$  (having computed  $W$  at line 13.)

### 3.1 Empirical Data

Table 3.1 gives parameters for 10 interpolants which were interpolated by our implementation in C++ of Algorithms 1 and 2. All of these interpolants are

calculated by taking the determinant of a maximal minor of some polynomial matrix; the size and rank of this matrix are given in the "Matrix Size" and "Rank" columns. The "# Terms" and "Degree" columns give the size and total degree of the target polynomial.

Table 3.1 compares the two methods of univariate interpolation for the problems in Table 3.1. Our implementation uses the  $O(n^2)$  algorithm to solve transposed Vandermonde systems of size up to 1500; for larger systems the  $O(n(\log n)^2)$  algorithm of [KL88] is superior. Newton interpolation, which is also  $O(n^2)$ , has a smaller leading constant and therefore needs less time than Vandermonde-style univariate interpolation for "small" polynomials (of degree  $\leq 1500$ ). This is why, for interpolants that prove to be dense in individual variables (problems 1, 2, 3, 5, and 9) the running time for Algorithm 2 is slightly longer than that of Algorithm 1.

However, for interpolants which are not dense in individual variables, the evaluations saved by the Vandermonde approach can yield substantial improvements in running time. For example, problem 8, where the target polynomial is both very large and very sparse, is solved nearly 3 times faster by Algorithm 2.



**Algorithm 3:** Zippel's Algorithm, generalized for multivariate images**Data:**  $\mathcal{Q}$  a field,  $I(x_1 \dots x_n)$  an expression in the variables  $x_1 \dots x_n$ **Result:**  $\forall a_1 \dots a_n \in \mathcal{Q}, P(a_1 \dots a_n) \in \mathcal{Q}[x_1 \dots x_n] = I(a_1 \dots a_n)$ 

```

1 for  $j = 1 \dots n$  do
2    $d_j \leftarrow \text{degree}(I, x_j)$ ;
3    $p_j \leftarrow$  a prime  $< |\mathcal{Q}|$ ,  $\neq p_i, i < j$ ;
4    $b_j \leftarrow \text{random}(\mathcal{Q})$ ;
5 SubsetImage( $\bar{x} \subset \{x_1 \dots x_n\}$ ): begin
6   if  $|\bar{x}| = 1$  then
7      $\bar{x} \equiv \{x_j\}$ 
8     for  $i = 0 \dots d_j$  do
9        $y_i \leftarrow \text{random}(\mathcal{Q})$ ;
10       $f_i \leftarrow I(b_1 \dots b_{j-1}, y_i, b_{j+1} \dots b_n)$ ;
11    return UnivarInterp( $(y_0, f_0) \dots (y_{d_j}, f_{d_j})$ );
12  else
13    Partition  $\bar{x}$  into  $\bar{x}_L \neq \emptyset, \bar{x}_R \neq \emptyset$ ;
14     $P_L \leftarrow \text{SubsetImage}(\bar{x}_L)$ ;
15     $\bar{p}_L \equiv \{p_i \mid x_i \in \bar{x}_L\}$ 
16     $(M_L, V_L) \leftarrow \text{Vandermonde}(P_L, \bar{p}_L)$ ;
17     $P_R \leftarrow \text{SubsetImage}(\bar{x}_R)$ ;
18     $\bar{p}_R \equiv \{p_i \mid x_i \in \bar{x}_R\}$ 
19     $(M_R, V_R) \leftarrow \text{Vandermonde}(P_R, \bar{p}_R)$ ;
20     $\bar{x}_B \equiv \{x_1 \dots x_n\} - \bar{x}, \bar{b}_B \equiv \{b_i \mid x_i \in \bar{x}_B\}$ 
21     $E \leftarrow \text{Matrix}(1 \dots |M_L|, 1 \dots |M_R|)$ ;
22    for  $l = 1 \dots |M_L|$  do
23      for  $r = 1 \dots |M_R|$  do
24         $E_{lr} \leftarrow I(\bar{p}_L^r, \bar{p}_R^l, \bar{b}_B)$ ;
25     $C \leftarrow V_L^{-1} \times E \times (V_R^{-1})^\top$ ;
26    return  $M_L^\top \times C \times M_R$ ;
27 end
28  $P \leftarrow \text{SubsetImage}(\{x_1 \dots x_n\})$ ;

```

Table 2: Comparing Univariate Interpolation Methods

#	Newton		Vandermonde		% Evals of Newton	
	# Evals	Run Time, s	# Evals	Run Time, s	actual	potential <sup>2</sup>
1	48312	97.22	48303	99.7	99.98	88.89
2	1474540	4499.57	1474540	4533.79	100.0	51.28
3	138259	291.11	138259	292.06	100.0	100.0
4	16173	25.28	15385	24.12	95.13	18.08
5	26474	64.31	26474	64.53	100.0	100.0
6	25188	39.57	18184	28.57	72.19	30.37
7	4367760	14275.9	4183964	14216.1	95.79	90.90
8	916060	538126	339692	198071	37.08	13.96
9	421328	12525.1	421328	12505.9	100.0	51.51
10	6487	9.16	4751	6.68	73.24	55.56

## 4 Using Interpolant Images in Several Variables

There is a change of perspective involved in the switch from interpolating the new variable by Newton's method to doing so by solving a transposed Vandermonde system in Zippel's algorithm. Previously, each stage of Zippel's algorithm extended the stage polynomial  $P_k$  by replacing each of its coefficients with a polynomial in  $x_{k+1}$ . Switching to transposed Vandermonde systems for interpolation means that the coefficients of  $P_k$  are now being replaced by copies of the image of the interpolant  $P$  in  $x_{k+1}$ . The altered algorithm, that is, is not so much extending a single image by a variable, as merging two images together, one of which happens to be univariate.

---

<sup>1</sup>This is  $\prod_{i=1}^{\#Vars} t_i / (d_i + 1)$ , where  $t_i$  is the size of the problem's image in variable  $i$  and  $d_i$  is its degree in that variable.

The technique for interpolating a polynomial by solving a transposed Vandermonde system, as explained in the previous section, works just as well for a multivariate polynomial as for a univariate one. In the general case, the size of the system to be solved, which equals the number of possible monomials, is  $O(d^n)$  where  $d$  is the highest degree in any variable and  $n$  is the number of variables, so a direct use of the technique is not practical for multivariate interpolants. But we can use the technique in the altered algorithm, taking two multivariate images of the interpolant, and replacing the coefficients of the first with copies of the second. If the second image is sparse, so that most of the possible monomials have already been eliminated, the transposed Vandermonde systems will remain small enough to solve in a reasonable time.

#### 4.1 The Algorithm

Suppose that the variables  $x_1 \dots x_n$  of an interpolant  $I$  are partitioned into three subsets  $\overline{x_L}$ ,  $\overline{x_R}$  and  $\overline{x_B}$ , and that we have calculated the image  $P_L$  of  $I$  in  $\overline{x_L}$ , and the image  $P_R$  of  $I$  in  $\overline{x_R}$ . Then

$$P_L \equiv \sum_{l=1}^{t_L} c_l M_{Ll} \quad \text{where} \quad c_l \in \mathcal{Q}, M_{Ll} = \prod_{y \in x_L} y^{e_y},$$

$$P_R \equiv \sum_{r=1}^{t_R} c_r M_{Rr} \quad \text{where} \quad c_r \in \mathcal{Q}, M_{Rr} = \prod_{y \in x_R} y^{e_y},$$

and we wish to calculate

$$P_N = \sum_{l=1}^{t_L} f_{Ll}(\overline{x_R}) M_{Ll} = \sum_{l=1}^{t_L} \left( \sum_{r=1}^{t_R} c_{l,r} M_{Rr} \right) M_{Ll}.$$

As in Section 2, we substitute for the variables  $\overline{x_L}$  in  $M_{L1} \dots M_{Lt_L}$  the primes  $\overline{p_L}$  and obtain a transposed Vandermonde system with the matrix

$$V_L = \begin{bmatrix} M_{L1}(\overline{p_L})^0 & \dots & M_{Lt_L}(\overline{p_L})^0 \\ M_{L1}(\overline{p_L})^1 & \dots & M_{Lt_L}(\overline{p_L})^1 \\ \vdots & \ddots & \vdots \\ M_{L1}(\overline{p_L})^{t_L-1} & \dots & M_{Lt_L}(\overline{p_L})^{t_L-1} \end{bmatrix}.$$

And, as in Section 3, we substitute for the variables  $\overline{x_R}$  in  $M_{R1} \dots M_{Rt_R}$  the primes  $\overline{p_R}$  to obtain another transposed Vandermonde matrix

$$V_R = \begin{bmatrix} M_{R1}(\overline{p_R})^0 & \dots & M_{Rt_R}(\overline{p_R})^0 \\ M_{R1}(\overline{p_R})^1 & \dots & M_{Rt_R}(\overline{p_R})^1 \\ \vdots & \ddots & \vdots \\ M_{R1}(\overline{p_R})^{t_R-1} & \dots & M_{Rt_R}(\overline{p_R})^{t_R-1} \end{bmatrix}.$$

Finally, we substitute powers of  $\overline{p_L}$  and  $\overline{p_R}$  into  $I$  to obtain the matrix

$$E = \begin{bmatrix} I(\overline{p_L}^0, \overline{p_R}^0, \overline{b_B}) & \dots & I(\overline{p_L}^0, \overline{p_R}^{t_R-1}, \overline{b_B}) \\ I(\overline{p_L}^1, \overline{p_R}^0, \overline{b_B}) & \dots & I(\overline{p_L}^1, \overline{p_R}^{t_R-1}, \overline{b_B}) \\ \vdots & \ddots & \vdots \\ I(\overline{p_L}^{t_L-1}, \overline{p_R}^0, \overline{b_B}) & \dots & I(\overline{p_L}^{t_L-1}, \overline{p_R}^{t_R}, \overline{b_B}) \end{bmatrix}$$

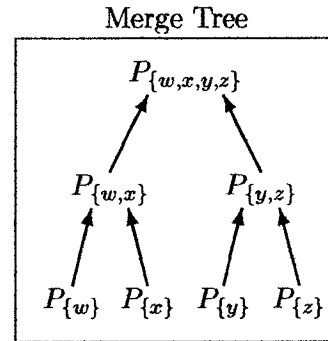
(where  $\overline{b_B}$  are anchor values, substituted uniformly for  $\overline{x_B}$ .)

Then the matrix equation  $V_L \times Z = E$  solves for the values of  $f_{L1}(\overline{x_R}) \dots f_{Lt_L}(\overline{x_R})$ , and the matrix equation  $V_R \times C^\top = Z^\top$  solves for the coefficients of  $f_{L1} \dots f_{Lt_L}$ , which are the coefficients of  $P_N$ . Substituting the latter equation into the former gives  $V_L \times C \times V_R^\top = E$  – the equation solved at line 25 of Algorithm 3.

Therefore, to interpolate  $I(x_1 \dots x_n)$  we fix primes  $p_1 \dots p_n$  and anchors  $b_1 \dots b_n$ , then calculate its image in all the variables  $x_1 \dots x_n$ ; and we calculate the image of  $I$  in a subset of  $x_1 \dots x_n$  by partitioning it into two further subsets  $\overline{x_L}$  and  $\overline{x_R}$ , recursively finding the images  $P_L, P_R$  of  $I$  in those subsets, and merging them as explained above. The base case for the recursion is subsets containing 1 variable, for which we use any univariate interpolation algorithm. Another view of the algorithm is as a binary tree, in which each leaf represents the univariate interpolation of 1 variable, each internal node represents the merging of its children; the root then computes the image in all the variables, a.k.a. the target polynomial.

## 4.2 Example

We will use again the interpolant from Section 2.2,  $I(w, x, y, z) = w^2x^3yz - 3wx^3yz^2 - w^2xy^2z + wx^3z^2 - 5xy^2z^2 + 2x^3z^2 + 2wx^3 - wx + 2z^2 - 4w$ , with the same primes (2, 3, 5, 7) and anchors (11, 13, 17, 19). There are 4 univariate interpolations (which are not shown) and 3 merges to perform.



$$\{\mathbf{w}\} \cup \{\mathbf{x}\} \rightarrow \{\mathbf{w}, \mathbf{x}\}$$

$$P_{\{w\}} = 638248w^2 - 39651473w - 5194429, \text{ and } P_{\{x\}} = -159445x^3 - 1186067x +$$

678; we wish to find

$$P_{\{w,x\}} = (C_{2,2}x^3 - C_{1,2}x + C_{0,2})w^2 + (C_{2,1}x^3 - C_{1,1}x + C_{0,1})w + (C_{2,0}x^3 - C_{1,0}x + C_{0,0}).$$

Solving the matrix equation  $V_{\{w\}} \times C \times V_{\{x\}}^\top = E$ , where

$$V_{\{w\}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^1 & 2^2 \\ 1^2 & (2^1)^2 & (2^2)^2 \end{bmatrix},$$

$$V_{\{x\}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3^1 & 3^3 \\ 1^2 & (3^1)^2 & (3^3)^2 \end{bmatrix},$$

and

$$E = \begin{bmatrix} I(2^0, 3^0, 17, 19) & I(2^0, 3^1, 17, 19) & I(2^0, 3^2, 17, 19) \\ I(2^1, 3^0, 17, 19) & I(2^1, 3^1, 17, 19) & I(2^1, 3^2, 17, 19) \\ I(2^2, 3^0, 17, 19) & I(2^2, 3^1, 17, 19) & I(2^2, 3^2, 17, 19) \end{bmatrix},$$

gives

$$C = \begin{bmatrix} 722 & -521645 & 722 \\ -4 & -1 & -18048 \\ 0 & -5491 & 323 \end{bmatrix},$$

so

$$\begin{aligned} P_{\{w,x\}} &= [1, w, w^2] \cdot C \cdot [1, x, x^3]^\top \\ &= 323w^2x^3 - 18048wx^3 - 5491w^2x + 722x^3 - wx - 4w - 521645x + 722. \end{aligned}$$

$$\{\mathbf{y}\} \cup \{\mathbf{z}\} \rightarrow \{\mathbf{y}, \mathbf{z}\}$$

$$P_{\{y\}} = -53352y^2 - 21121958y + 10359390, \text{ and } P_{\{z\}} = -1222739z^2 + 4064632z +$$

48147; we wish to find

$$P_{\{y,z\}} = (C_{2,2}z^2 - C_{1,2}z + C_{0,2})y^2 + (C_{2,1}z^2 - C_{1,1}z + C_{0,1})y + (C_{2,0}z^2 - C_{1,0}z + C_{0,0}).$$

Solving the matrix equation  $V_{\{y\}} \times C \times V_{\{z\}}^\top = E$ , where

$$V_{\{y\}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 5^1 & 5^2 \\ 1^2 & (5^1)^2 & (5^2)^2 \end{bmatrix},$$

$$V_{\{z\}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 7^1 & 7^2 \\ 1^2 & (7^1)^2 & (7^2)^2 \end{bmatrix},$$

and

$$E = \begin{bmatrix} I(11, 13, 5^0, 7^0) & I(11, 13, 5^0, 7^1) & I(11, 13, 5^0, 7^2) \\ I(11, 13, 5^1, 7^0) & I(11, 13, 5^1, 7^1) & I(11, 13, 5^1, 7^2) \\ I(11, 13, 5^2, 7^0) & I(11, 13, 5^2, 7^1) & I(11, 13, 5^2, 7^2) \end{bmatrix},$$

gives

$$C = \begin{bmatrix} 48147 & 0 & 28563 \\ 0 & 265837 & -72501 \\ 0 & -1573 & -65 \end{bmatrix},$$

so

$$\begin{aligned} P_{\{y,z\}} &= [1, y, y^2] \cdot C \cdot [1, z, z^2]^\top \\ &= -65y^2z^2 - 1573y^2z - 72501yz^2 + 265837yz + 28563z^2 + 48147. \end{aligned}$$

$$\{\mathbf{w}, \mathbf{x}\} \cup \{\mathbf{y}, \mathbf{z}\} \rightarrow \{\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}\}$$

$$M_{\{w,x\}} = \{1, w, x, wx, w^2x, x^3, wx^3, w^2x^3\}, \text{ and } M_{\{y,z\}} = \{1, yz, z^2, y^2z, yz^2, y^2z^2\}.$$

$$V_{\{w,x\}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2^1 & 3^1 & 2^1 3^1 & 2^2 3^1 & 3^3 & 2^1 3^3 & 2^2 3^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & (2^1)^7 & (3^1)^7 & (2^1 3^1)^7 & (2^2 3^1)^7 & (3^3)^7 & (2^1 3^3)^7 & (2^2 3^3)^7 \end{bmatrix},$$

$$V_{\{y,z\}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 5^1 7^1 & 7^2 & 5^2 7^1 & 5^1 7^2 & 5^2 7^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & (5^1 7^1)^5 & (7^2)^5 & (5^2 7^1)^5 & (5^1 7^2)^5 & (5^2 7^2)^5 \end{bmatrix},$$

and

$$E = \begin{bmatrix} I(2^0, 3^0, 5^0, 7^0) & \cdots & I(2^0, 3^0, 5^5, 7^5) \\ \vdots & \ddots & \vdots \\ I(2^7, 3^7, 5^0, 7^0) & \cdots & I(2^7, 3^7, 5^5, 7^5) \end{bmatrix},$$

so

$$C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and

$$\begin{aligned} P_{\{w,x,y,z\}} &= [1, w, x, wx, w^2x, x^3, wx^3, w^2x^3] \cdot C \cdot [1, yz, z^2, y^2z, yz^2, y^2z^2]^\top \\ &= w^2x^3yz - 3wx^3yz^2 - w^2xy^2z + wx^3z^2 - 5xy^2z^2 + 2x^3z^2 + \\ &\quad 2wx^3 - wx + 2z^2 - 4w = I. \end{aligned}$$

### 4.3 Analysis

For an interpolant  $I$  of  $n$  variables, the algorithm calls **SubsetImage**  $2n - 1$  times, branching to line 7  $n$  times and to line 13  $n - 1$  times. From line 7 **SubsetImage** performs up to  $d + 1$  evaluations of  $I$  (where  $d = \max\{d_1 \dots d_n\}$ ) and calls **UnivarInterp** once; thus this branch of **SubsetImage** contributes  $n(d + 1)$  evaluations of  $I$ , and  $O(nd^2)$  additional arithmetic operations. From line 13 **SubsetImage** performs  $|M_L| |M_R|$  evaluations of  $I$  to construct  $E$ . Then it calculates  $V_L^{-1} \times E \times (V_R^\top)^{-1}$ ; this calls for  $|M_R|$  solutions of  $V_L$  and  $|M_L|$  solutions of  $V_R$ . With Kaltofen & Yagati's method



one solution of a  $t \times t$  Vandermonde matrix is found in  $O(t \log^2 t)$  arithmetic operations. Since at every call to **SubsetImage**  $|M_L|$  and  $|M_R|$  are no greater than  $t$  the number of terms in the target polynomial, the branch at line 13 thus contributes  $(n-1)O(t^2)$  evaluations and  $(n-1)O(t^2 \log^2 t)$  additional arithmetic operations. The algorithm's complexity is therefore

$$O(n((d+t^2)O(I) + d^2 + t^2 \log^2 t)),$$

where  $O(I)$  is the complexity of evaluating  $I$  once.

As with Zippel's algorithm, there are two possible sources of error in this algorithm. A zero coefficient in an image of the interpolant may be not the value of an identically zero sub-polynomial in the target, but of a sub-polynomial that vanishes at the anchor values. At each call to **SubsetImage**, there are no more than  $t$  polynomials, in less than  $n$  variables, each of degree less than  $d$ , which must not be 0 at the anchor; and **SubsetImage** is called  $2n-1$  times. By Theorem 2.1 the probability of a zero polynomial is

$$O\left(\frac{n^2 dt}{q}\right),$$

of the same order as Zippel's' algorithm.

The other source of error is the possibility that a Vandermonde matrix ( $V_L$  or  $V_R$ ) will be singular. The branch at line 7 constructs no Vandermonde matrices. The branch at line 13 constructs 2 matrices, and by Theorem 2.2

the probability that either is singular is no more than

$$\frac{dT(T-1)}{2(|\mathcal{Q}|-1)}.$$

As **SubsetImage** enters this branch  $n-1$  times, and the size of either matrix is no more than  $t$ , the probability of a failure from this source is no more than

$$O\left(\frac{ndt^2}{|\mathcal{Q}|}\right),$$

also of the same order as Zippel's algorithm.

## 5 Conclusions

The probability of error for Algorithm 3 is of the same order as Algorithm 1's; but the time complexities for Algorithm 3 exceeds that of Algorithm 1 by a factor of  $t/d$ . Since in almost all interpolants  $t \gg d$ , careless partitioning at line 13 will make Algorithm 3 considerably slower than Algorithm 1, by inflating the number of evaluations. In our implementation, when variables were partitioned at random, the call to **SubsetImage** at the root of the tree nearly always needed  $O(t^2)$  evaluations, and that single call (not including its children in the tree) consistently accounted for 99% of the algorithm's running time.

A simple way for Algorithm 3 to recover the performance of Algorithms 1 and 2 is to put, at line 13, only one variable into  $\overline{x_R}$ . This essentially duplicates the variable-by-variable interpolation of Zippel's algorithm. When

the interpolant is a genuine "black box", and nothing is known about the structure of the target polynomial, this is likely to be the best partition policy.

If, however, there is a subset of variables  $\bar{x}$  for which the interpolant's image in  $\bar{x}$  is known to have  $O(d)$  terms (which is not impossible) then taking  $\overline{x_R} = \bar{x}$  as soon as possible gives a better policy than Zippel's. For finding the image in  $\bar{x}$  calls for  $O(|\bar{x}| d^2)$  evaluations, and then the interpolation can be finished with  $O(dt)$  evaluations; whereas interpolating  $\bar{x}$  variable by variable calls for  $O(|\bar{x}| dt)$  evaluations. In effect the variables in  $\bar{x}$  can be interpolated all at once in no more time than a single variable's interpolation requires.

One direction of future research, then, would be analysis of interpolants to discover good ways to partition their variables, thereby exploiting sparsity in their multivariate images.

## 6 Related Work

Wen-shin Lee [Lee01] gives two refinements of Zippel's algorithm, named *temporary and permanent pruning*, which decrease the number of evaluations needed for interpolation. Temporary pruning uses an adaptive method for univariate interpolation, and instead of generating the whole matrix  $E$  of evaluations, performs only enough evaluations in one row of  $E$  to interpolate the corresponding entry of the array of coefficient polynomials  $F$ . Permanent

pruning alters the interpolant  $I$  by substituting for each variable  $x_1 \dots x_n$  the product  $zx_1 \dots zx_n$  of that variable with a new variable  $z$ . The effect of this is that in each term of the altered interpolant  $I'$  the degree of  $z$  will equal the total degree of the term in the original interpolant. The image of  $I'$  in  $z$  therefore separates terms of different total degrees in  $I$  into different sub-polynomials.

Temporary pruning is, unfortunately, not compatible with Vandermonde univariate interpolation. Newton interpolation generates successive approximations to the target polynomial, of successively greater degree, and thus can be adaptive to the size of its target polynomial; [Lee01] gives the method. Vandermonde systems, however, must be solved all at once.

Permanent pruning is compatible with Algorithms 2 and 3, and our implementation of these algorithms has an option to introduce a homogenizing variable. Indeed, due to the recursive structure of Algorithm 3, it would be possible to use permanent pruning on *subsets* of an interpolant's variables – which may prove useful for resultants, which are often "multi-homogenous". (A homogenous multivariate polynomial is one in which every term has the same total degree; a multi-homogenous polynomial is homogenous, and the image of the polynomial in some subset of its variables is also a homogenous polynomial.)

## References

- [BOT88] Michael Ben-Or and Prason Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing (New York, NY, USA), ACM Press, 1988, pp. 301–309.
- [Cht03] A.D. Chtcherba, *A new Sylvester-type Resultant Method based on the Dixon-Bézout Formulation*, Ph.D. thesis, Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA, Jul 2003.
- [Dix08] A.L. Dixon, *The eliminant of three quantics in two independent variables*, Proceedings of London Mathematical society (1908), no. 6, 3–27.
- [Kap96] Deepak Kapur, *Automated geometric reasoning: Dixon resultants, gröbner bases, and characteristic sets.*, Automated Deduction in Geometry, 1996, pp. 1–36.
- [KL88] Erich Kaltofen and Yagati N. Lakshman, *Improved sparse multivariate polynomial interpolation algorithms.*, Proceedings of Symbolic and Algebraic Computation, International Symposium, IS-

- SAC'88 (Rome, Italy) (Patrizia M. Gianni, ed.), 1988, pp. 467–474.
- [KL03] Erich Kaltofen and Wen-Shin Lee, *Early termination in sparse interpolation algorithms.*, J. Symb. Comput. **36** (2003), no. 3-4, 365–400.
- [KLL00] Erich Kaltofen, Wen-Shin Lee, and A. Lobo, *Early termination in ben-or/tiwari sparse interpolation and a hybrid of zippel's algorithm.*, ISSAC, 2000, pp. 192–201.
- [KSY94] D. Kapur, T. Saxena, and L. Yang, *Algebraic and geometric reasoning using the Dixon resultants*, ACM ISSAC 94 (Oxford, England), July 1994, pp. 99–107.
- [Lee01] Wen-Shin Lee, *Early termination strategies in sparse interpolation algorithms*, Ph.D. thesis, North Carolina State University, Berkeley, 2001.
- [Mac02] F.S. Macaulay, *On some formula in elimination*, Proceedings of London Mathematical society (1902), 3–27.
- [Mac16] ———, *The algebraic theory of modular systems*, Cambridge Tracts in Math. and Math. Phys. **19** (1916).

- [Man92] D. Manocha, *Algebraic and numeric techniques for modeling and robotics*, Ph.D. thesis, Computer Science Division, University of California, Berkeley, 1992.
- [PFTV90] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical recipes: The art of scientific computing*, Cambridge University Press, 1990.
- [Syl53] J.J Sylvester, *On a theory of syzygetic relations of two rational integral functions, comprising an application to the theory of sturm's functions, and that of the greatest algebraic common measure*, Philosophical Trans. **143** (1853), 407–548.
- [Zip90] Richard Zippel, *Interpolating polynomials from their values.*, J. Symb. Comput. **9** (1990), no. 3, 375–403.
- [Zip93] ———, *Effective polynomial computation*, Kluwer Academic Publishers, Boston, MA, 1993.

## **Vita**

Michael David Brazier was born on January 20, 1972, in Columbus, Ohio. He graduated from Edinburg High School (Edinburg, Texas) in 1989, and attained the Bachelor of Science degree in Mathematics from the University of Texas-Pan American in 1996. His current mailing address is

921 Tori Lane

Edinburg, TX 78539