Sparse Polynomial Interpolation via Discrete Logs

Adriano Arce Michael Monagan Hao Zhuang

The Multivariate GCD Problem

Fix a prime p and some $n \in \mathbb{N}$ and choose any multivariate polynomials $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$. Then *the multivariate GCD problem* is to efficiently compute $G = \gcd(A, B) \pmod p$. It turns out that the fastest known algorithms for solving this problem each use the same general strategy: compute several univariate images of G in $\mathbb{Z}_p[x_0]$, then recover G via sparse interpolation.

Sparse Polynomials

In practice, multivariate polynomials are usually sparse. More precisely, let $d = \deg G$ be the total degree of G and let $T = \#G$ be the number of nonzero terms in G. Then we say that G is *sparse* iff $T \ll \binom{n+d+1}{d}$, the maximum number of terms. For example, the following polynomial contains only $T = 5$ terms (which is much less than $\binom{6+10+1}{10} = 19448$) and thus is considered very sparse:

$$G = x_0^{10} + 7x_0^3 x_1 x_6^2 + 6x_0^3 x_5 + 8x_1 x_2 x_3^7 + 1$$

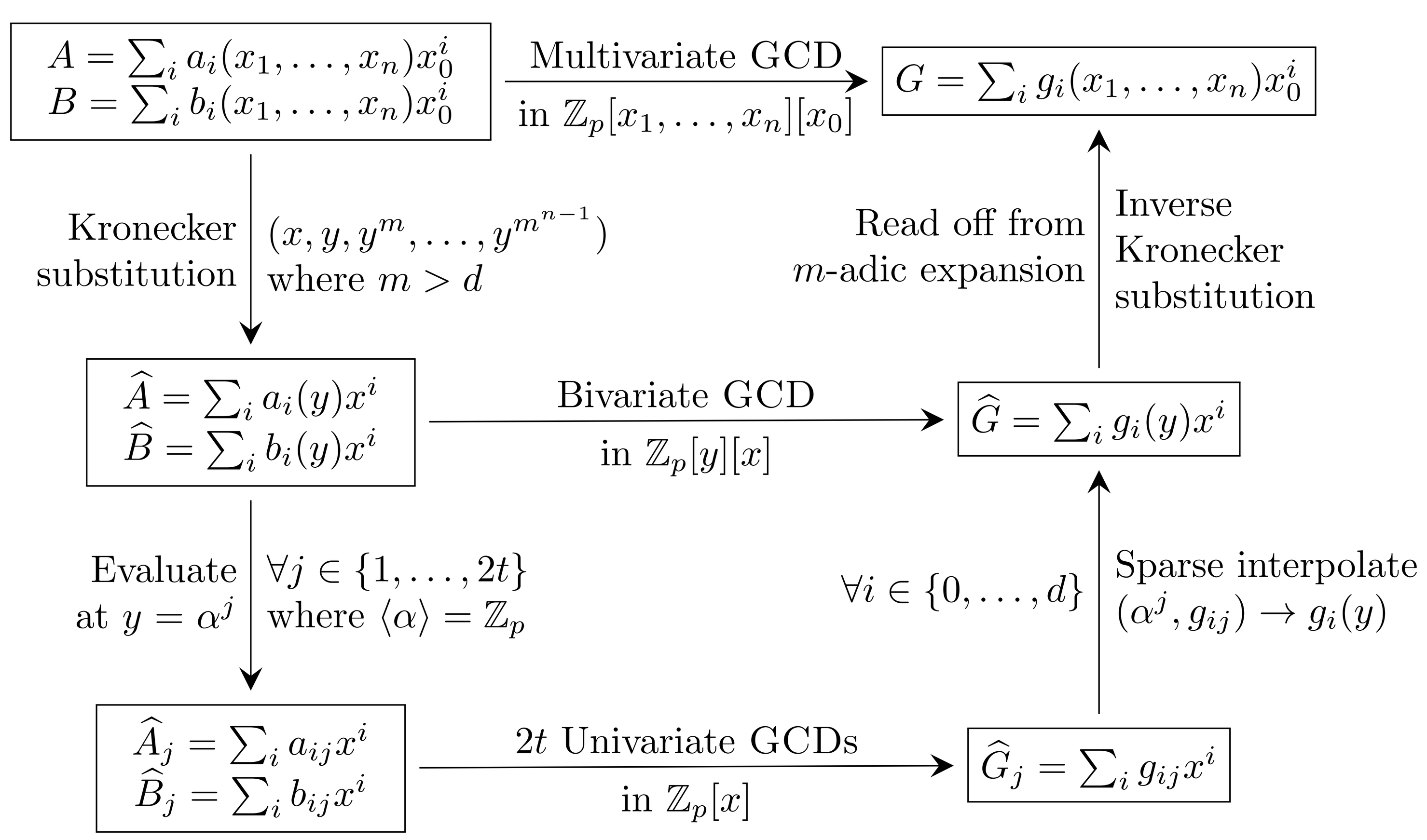
Previous Multivariate GCD Algorithms

Let $G = \sum_i g_i(x_1, \ldots, x_n) x_0^i$ and let $t_i = \#g_i$ be the number of terms in g_i and let $t = \max_i t_i$. Generally, we want to minimize the number of images required for interpolation since evaluations typically represent the bottleneck step. Below is a table of previous multivariate GCD algorithms:

Year	Author(s)	Randomness	# of Images
1971	Brown [3]	Deterministic	$O(d^n)$
1979	Zippel [6]	Probabilistic	$O(ndt)$
1988	Ben-Or/Tiwari [2]	Probabilistic	$O(t)$

We present a modified version of Ben-Or/Tiwari's algorithm [2] that also requires only $O(t)$ images. Unlike Ben-Or/Tiwari's algorithm however (which requires that we choose p to be bigger than p_n^d, where p_n is the n^{th} smallest prime), our approach only requires that $p > d^n$.

Overview of our Multivariate GCD Algorithm



The Kronecker Substitution

Given any $F(x_0, x_1, \ldots, x_n) \in \mathbb{Z}_p[x_0, x_1, \ldots, x_n]$, fix some $m > \deg F$. Then we define *the Kronecker substitution* of F to be:

$$\widehat{F} = F(x, y, y^m, y^{m^2}, \ldots, y^{m^{n-1}})$$

Notice that the Kronecker substitution allows us to map a GCD computation modulo p in $n+1$ variables into just 2 variables. Furthermore, observe that we can recover F from $\widehat{F}$ (since $m > \deg F$). Unfortunately, there are certain values of m that represent "unlucky" Kronecker substitutions. For example, consider:

$$A = x_0^2 - x_1^2 x_2 \qquad B = x_0^3 + x_1 x_2^2 \qquad m = 4$$

Notice that $\widehat{G} = \gcd(\widehat{A}, \widehat{B}) = \gcd(x^2 - y^6, x^3 + y^9) = x + y^3$ while the true GCD is $G = \gcd(A, B) = 1$. In this case, it is impossible to recover G from $\widehat{G}$. Fortunately, we can prove that there are only finitely many $m > d$ for which the Kronecker substitution fails in this way.

The Evaluation Points

Let:

$$\widehat{F} = \underbrace{\sum_{i=0}^{\deg_x \widehat{F}} f_i(y) x^i}_{\text{dense format}} = \underbrace{\sum_{i=1}^{s} u_i x^{v_i} M_i(y)}_{\text{sparse format}}$$

where $s = \#\widehat{F}$ is the number of terms in $\widehat{F}$ and $M_i(y) = y^{w_i}$ are called the *monomials*. We want to evaluate $\widehat{F}$ at $y = \alpha^j$ for each $j \in \{1, \ldots, 2t\}$.

At first, we did this by evaluating the monomials one at a time using simple binary powering. Since $\deg_y \widehat{F} < (d+1)^n$, this required a total of $O(stn \log d)$ multiplications in $\mathbb{Z}_p$. However, since evaluation turned out to be the bottleneck of the entire GCD algorithm, we decided to use a different technique.

Notice that $M_i(\alpha^j) = (\alpha^j)^{w_i} = (\alpha^{w_i})^j = (M_i(\alpha))^j$. Hence, if we compute $\Gamma = [M_1(\alpha), \ldots, M_s(\alpha)] \in \mathbb{Z}_p^s$ in $O(s \log d)$ multiplications, then we can compute the next $\widehat{F}(x, \alpha^{j+1})$ from $\widehat{F}(x, \alpha^j)$ using only s multiplications so that this step requires a total of $O(s \log d + st)$ multiplications in $\mathbb{Z}_p$. Note however that this makes the evaluations serial. To parallelize this for N cores, we use a baby-step giant-step algorithm.

Sparse Interpolation via $\Lambda_i(z)$

Given the points (α^j, g_{ij}) for $j \in \{1, \ldots, 2t\}$, we want to interpolate the sparse polynomial $g_i(y) = \sum_{k=1}^{t_i} c_k M_k(y)$ where $t_i = \#g_i$ and $c_k \in \mathbb{Z}_p^*$ and $M_k(y) = y^{d_k}$. That is, we seek each c_k and d_k. To this end, let $m_k = M_k(\alpha) = \alpha^{d_k}$ and consider the *linear generator* defined by:

$$\Lambda_i(z) = \prod_{k=1}^{t_i} (z - m_k) = z^{t_i} + \sum_{k=0}^{t_i - 1} \lambda_k z^k$$

We could obtain each λ_k from the (α^j, g_{ij}) by solving a linear system in $O(t^3)$ arithmetic operations in $\mathbb{Z}_p$. Instead, we obtain the λ_k by using an extended Euclidean version of the Berlekamp-Massey algorithm [1], which only takes $O(t^2)$ operations. We then compute each of the roots m_k via Rabin's Las Vegas algorithm [5] in $O(t^2 \log p)$ operations.

Discrete Logarithms

For each of the t_i roots $m_k = \alpha^{d_k}$, we want to efficiently compute the *discrete logarithm* given by $d_k = \log_\alpha m_k$ in $\mathbb{Z}_p$. In general, this is very difficult (many people suspect that it is NP-hard, and the security of the Diffie-Hellman key exchange protocol from cryptography relies on this). However, for Fourier primes of the form $p = 2^r q + 1$ with q sufficiently small, the problem is no longer intractable.

By using the Pohlig-Hellman algorithm [4], we can compute each d_k using only $O(\sqrt{q} + r \log r)$ operations in the cyclic group $\mathbb{Z}_p^*$. This choice for p also means that we can apply the Fast Fourier Transform inside $\mathbb{Z}_p$ to accelerate Rabin's algorithm from $O(t^2 \log p)$ to $O(t \log t \log p)$. Note that to ensure that the m_k are distinct, we require that $p > \deg_y \widehat{G}$. We may use $\deg_y \widehat{G} \le \min\{\deg_y \widehat{A}, \deg_y \widehat{B}\}$.

Shifted Transposed Vandermonde Systems

To solve for the unknown coefficients c_k we solve the shifted transposed Vandermonde system

$$\begin{bmatrix} m_1 & m_2 & \cdots & m_t \\ m_1^2 & m_2^2 & \cdots & m_t^2 \\ \vdots & \vdots & \ddots & \vdots \\ m_1^t & m_2^t & \cdots & m_t^t \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} g_{i1} \\ g_{i2} \\ \vdots \\ g_{it} \end{bmatrix}$$

By taking advantage of its structure, we can accomplish this by using only $O(t^2)$ arithmetic operations in $\mathbb{Z}_p$ and $O(t)$ space (see Zippel [6]).

Parallel Implementation and Benchmarks

We have implemented our algorithm in Cilk C, a parallel extension of C which has been adopted by Intel for the Intel C compiler. We have parallelized the evaluations, and we interpolate the coefficients $g_i(y)$ of $\widehat{G}$ in parallel. Since our algorithm requires that $p > d^n$, we have implemented our algorithm for 31-bit and 63-bit primes, and we are working on a 127-bit prime implementation.

To assess our algorithm's performance, we compared it with the implementation of Zippel's algorithm in Maple and a Hensel Lifting algorithm in Magma. The following timings are in CPU seconds:

		3 variables				6 variables			
$\#G$	d	1 core	8 cores	Maple	Magma	1 core	8 cores	Maple	Magma
1000	10	0.062	0.015	0.076	0.08	1.306	0.232	35.61	3.38
2000	20	0.238	0.048	0.385	0.89	2.585	0.488	166.55	137.76
5000	50	1.231	0.270	5.174	20.00	6.623	1.239	1338.18	8527.85
10000	100	3.628	0.770	72.461	228.84	13.239	2.459	5310.27	–
20000	200	7.094	1.666	693.088	3003.23	26.610	4.915	–	–

References

[1] N. B. Atti, G. M. Diaz-Toca, and H. Lombardi. The Berlekamp-Massey Algorithm revisited. *Communication and Computing*, AAECC **17**(1), 75-82, 2006.

[2] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pp. 301-309, 1988.

[3] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* **18** (1971), 478-504.

[4] S. C. Pohlig and M. E. Hellman. An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance. *IEEE Trans. on Inf. Theory* **24**(1), pp. 106-110, 1978.

[5] M. O. Rabin. Probabilistic Algorithms in Finite Fields. *SIAM J. Computing*, **9**(3), 273-280, 1980.

[6] R. Zippel. Interpolating Polynomials from their Values. *J. Symbolic Comput.* **9**, 3 (1990), 375-403.