Polynomial GCD Algorithms

Ativ Joshi
Summer Intern
IIT Gandhinagar

August 16, 2017

Introduction

The GCD of two non zero polynomials exists iff coefficients of the polynomials belong to a unique factorization domain. GCD of polynomials is unique up to the multiplication by an invertible constant (a unit), i.e. there is a set of GCDs where each one is a unit multiple of the other.

Euclid's algorithm can be used to calculate GCD of polynomials with coefficients in a field by constructing Polynomial Remainder Sequences (PRS).

For polynomials in $\mathbb{Z}[x]$, one approach is to work in $\mathbb{Q}[x]$ in order to apply Euclid's algorithm as $\mathbb{Z}$ is not a field. For multivariate polynomials we need to work in $\mathbb{Z}(x_1, ..., x_{n-1})[x_n]$.Since coefficients are rational functions recursive application of GCD algo is necessary to perform coefficient arithmetic.The second approach for polynomials in $\mathbb{Z}[x]$ is to build a sequence of pseudo remainders using pseudo division rather than quotient field polynomial division. In this approach, all the arithmetic operations would be done in the domain of $\mathbb{Z}[x]$.

The problem with both of the above approaches is the growth in size of the coefficients. Particularly, in the case of pseudo remainder sequences, the growth is exponential.One solution to prevent the exponential growth is to remove the content at each step. But this would require a significant number of GCD calculation which is not at all feasible, especially in the case of multivariate polynomials. To avoid calculating GCD, reduced PRS or sub-resultant PRS algorithms can be used but the growth in size of coefficients is still linear.A better way to deal with this problem is to use modular GCD algorithms.

Modular GCD Algorithm [1]

We can map the problem of finding the GCD to a simpler domain via multiple homomorphisms which avoid the issues with coefficient growth.

Lemma 1. *Let R and R' be UFDs with $\phi : R \to R'$ a homomorphism of rings. This induces a natural homomorphism also denoted by ϕ from $R[x]$ to $R'[x]$. suppose $A(x), B(x) \in R[x]$ and $C(x) = GCD(A(x), B(x))$ with $\phi(lcoeff(C(x))) \neq 0$. Then*

$$deg(GCD(\phi(A(x)), \phi(B(x)))) \geq deg(GCD(A(x), B(x)))$$

A homomorphism is said to be 'unlucky' if $deg(GCD(\phi(A(x)), \phi(B(x)))) > deg(GCD(A(x), B(x)))$. But the number of unlucky homomorphisms is finite and small which can often be detected. After detection, the unlucky homomorphic image is neglected and new images are calculated.

We use two types of coefficient homomorphisms in case of polynomial GCDs: (i). modular homomorphism $\phi_m : \mathbb{Z} \to \mathbb{Z}_m$, which maps integers into remainder modulo m and (ii). evaluation homomorphism $\phi_{w-b} : R[w] \to R$ which maps a polynomial in variable w to its value at w=b.

The process of finding GCD contains the following three algorithms[1] :

MGCD - This algorithm reduces the multivariate integer GCD problem to a series of multivariate finite field GCD problems by applying modular homomorphisms. This algorithm uses PGCD to calculate GCD in a finite field.

PGCD - This algorithm reduces the k-variate finite field GCD problem to a series of (k - 1)-variate finite field GCD problems by applying evaluation homomorphisms. This algorithm is used recursively and uses UGCD as the base case of recursion.

UGCD - This algorithm calculates GCD of univariate polynomials in a finite field.

The problem with these algorithms is that the number of domains required is large and exponential in number of variables. The number of domains used is larger than necessary, especially in the case of sparse polynomials. This usually occurs in the case of multivariate polynomials where number of non zero terms are smaller than the total number of terms possible.

Sparse Modular GCD Algorithms

Zippel's Algorithm [3]

This is a probabilistic algorithm which uses a combination of dense and sparse interpolation to construct the GCD of polynomial. It is based on the observation that evaluating a polynomial at a random point will almost never yield zero (Schwartz-Zippel lemma) [3]. So, if a coefficient is determined to be zero somewhere in the interpolation process, the algorithm assumes it to be zero everywhere.

Zippel's Algorithm requires a bound on partial degree in each variable d as an input. We substitute randomly chosen integers (starting point) for all but one variable. The solution is built up by interpolating for one variable at a time.The first polynomial produced involving a particular variable is done via dense interpolation. This polynomial is then used as a skeleton for a number of sparse interpolation. These sparse interpolations are used to introduce the next variable.After finding GCD modulo a prime, we treat it as a skeleton for finding the GCD modulo some other prime. The final polynomial is constructed using Chinese Remainder Algorithm.

The correctness algorithm depends on the accuracy of the skeletal polynomials. Skeletal polynomials will have missing terms if some coefficients are zero at the starting point. But this happens rarely as the total number of zeros are bounded by Schwartz-Zippel lemma, which states that if the integers in starting point are chosen from a subset $\mathbb{S}$ of a field, then probability of a coefficient being zero is bounded by $d/|\mathbb{S}|$, where d is the total degree of the coefficient.

The algorithm performs $O(ndt)$ evaluations for the interpolation process and runs in $O(ndt^3)$ time, where n is the number of variables, d is degree in each variable and t is number of terms in the final polynomial. As described in [4], Zippel's algorithm requires some modification if the GCD is not monic in the main variable, as it is difficult to scale univariate images of GCD in main variable consistently. This is termed as 'normalization problem' in [4].

Ben-Or/Tiwari's Algorithm [7]

This is a deterministic interpolation algorithm which interpolates for all the variables simultaneously unlike Zippel's algorithm where interpolation is done one variable at a time. The algorithm requires upper bound of $T \geq t$, where t is the number of terms in the final polynomial. But unlike Zippel, it does not require upper bound on partial degrees of the polynomial. For the interpolation purpose, the algorithm takes first n primes and uses them to construct $2T$ evaluation points $(2^i, 3^i, ..., p_n^i), 0 \leq i \leq 2T-1$, where n is the number of variables and T is the upper bound on the number of terms.

The algorithm can be divided into two parts. In the first part, we determine all the monomials of the polynomial by first constructing a linear generator $\lambda(z)$. This is done by solving a system of linear equations using Berlekamp-Masseyy Algorithm [8]. We then find the roots of this linear generator. The roots are actually the monomials evaluated at $(2, 3, ..., p_n)$. These monomials can be recovered by finding the prime decomposition of the roots. In the second part, we calculate the coefficients of each monomial by solving a transposed Vandermonde system of linear equations [9].

A drawback of Ben-Or/Tiwari's Algorithm is that it is inefficient to use this algorithm in a finite field as the prime required for the modulo operations have the length of O(D log n), where D is the total degree of polynomial.

Ben-Or/Tiwari with discrete logarithm [10]

This method is a modified form of Ben-Or/Tiwari algorithm such that the required size of prime is O(n log d) which is smaller than the original algorithm. Here, we pick a prime p of the form $p = q_1 q_2 ... q_n + 1$ satisfying $q_i > deg_{x_i} C$, where $deg_{x_i} C$ is the degree of x_i in the polynomial C which is to be interpolated.

Now, we find a primitive α generator of the field $\mathbb{Z}_p$ and generate $\omega_i = \alpha^{(p-1)/q_i}$. This algorithm replaces the primes $(2^i, 3^i, ..., p_n^i)$ used in the previous algorithm with $\omega_i, 1 \leq i \leq n$. Then, instead of prime decomposition, we find the monomials by calculating the discrete logarithms. Rest of the steps remain the same.

In practice, number of terms is not known in advance. But for a sufficiently large p, we can compute the linear generator and check its degree periodically. If the degree of $\lambda(z)$ remains constant, then we can say that the number of terms is t with high probability [10].

The Ben-Or/Tiwari is a deterministic algorithm, but there can still be unlucky evaluations. To solve this, instead of using $(2^i, 3^i, ..., p_n^i), 0 \leq i \leq 2t-1$, we use the points $s \leq i < s + 2t$. When we encounter unlucky evaluation, we shift this sequence. The resulting shifted transposed Vandermonde system can also be solved easily. But this type of modification is problematic for discrete log method as picking $q_i > 2t$ is difficult with t being unknown. This issue can be resolved using Kronecker Substitution as stated in [10].

Hu and Monagan's Parallel Sparse Polynomial GCD Algorithm [10]

Let $A = GA' = \Sigma_{i=0}^{dA} a_i x_0^i$, $B = GB' = \Sigma_{i=0}^{dB} b_i x_0^i$ and $G = \Sigma_{i=0}^{dG} c_i x_0^i$ where $a_i, b_i, c_i \in \mathbb{Z}[x_1, ..., x_n]$. We assume that GCD$(a_i) = 1$ and GCD$(b_i) = 1$. Let $\Gamma = GCD(LC(A), LC(B))$, where LC(A) denotes leading coefficient of A taken in x_0. Let $H = \Delta \times G$ and

$h_i = \Delta \times c_i$, so that $H = \Sigma_{i=0}^{dG} h_i x_0^i$.

The approach is to compute H modulo a sequence of primes p_1, p_2..... and recover the integer coefficients of H using Chinese Remainder Algorithm. After calculating the first image of H which is H mod p_1, the rest of the images will be calculated using Zippel's approach.

Kronecker Substitution

Kronecker Substitutions is used to map a multivariate polynomial into a bivariate polynomial. Kronecker substitution of a polynomial f in $D[x_0, x_1, ...x_n]$ is given by $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, ..., y^{r_1 r_2 \cdots r_{n-1}})$. If $d_i = deg_{x_i}(f)$ is the partial degrees of f for $1 \leq i \leq n$, then Kronecker substitution is invertible only if $r_i > d_i$ for $1 \leq i \leq n-1$.

After Kronecker substitution, we select the prime of the form $p = 2^k q + 1$ with q small. Then we find a random primitive generator α. Now we will compute several monic images $g_j = GCD(a_i, b_i)$ in $\mathbb{Z}_p[x]$, where $a_i = K(A)(x, \alpha^j)$ and $b_i = K(B)(x, \alpha^j)$ while taking care of the unlucky evaluations by shifting the sequence. Then we scale these images by $K(\Gamma)(\alpha^j)$. We run the Berlekamp-Massey algorithm on the scaled images and repeat till all the linear generators $(\lambda_i(z), 0 \leq i \leq d_0)$ corresponding to each degree in x. After creating $2t$ number of images, we then complete the interpolation process using the discrete logarithm method.

Note that the Δ is introduced due to the interpolation of scaled versions of the images. We cannot calculate gcd(K(A),K(B)) directly as the content would be exponential in number of variables. The important observation is that if we compute monic images $g_j = gcd(a_i, b_i)$, all the content is divided out and when we scale by $K(\Gamma)(\alpha^j)$ and interpolate y in k(H), we recover the divided content. Similar type of approach was used in Macsyma implementation of Zippel's algorithm [4]

The algorithm outputs H modulo p. This is a Monte-Carlo GCD algorithm and probabilities are involved at several stages.

The probability that some α selected from $\mathbb{Z}_p$ is unlucky is given by:

$$prob[\alpha \textit{ is bad or unlucky }] = \frac{degA\ degB + degA + degB}{p - degA - degB}$$

The maximum number of bad kronecker substitutions is $\sqrt{2}(n-1)[\sqrt{degA} + \sqrt{degB} + \sqrt{degAdegB}]$.

The BMA encounters the first zero discrepancy (i.e., the linear generator $\lambda_i(z)$ does not change) with probability at least

$$1 - \frac{t(t+1)(2t+1)deg(C)}{6|S|}$$

where S is the set of all possible evaluation points.

List of Algorithms

Algorithm	Year	Reference
Brown's Algorithm	1971	[11]
EZ-GCD	1973	[12]
Zippel's Algorithm	1979	[3]
EEZ-GCD Algorithm	1980	[13]
Ben-Or/Tiwari	1988	[7]
Linzip	2005	[4]
Ratzip	2005	[4]
Monagan/Javadi	2010	[14]
Monagan/Hu	2016	[10]

References

[1] Geddes, Keith O., Stephen R. Czapor, and George Labahn. Algorithms for computer algebra. Springer Science and Business Media,1992.

[2] D.E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms (second edition), Addison-Wesley (1981).

[3] Zippel, Richard. "Probabilistic algorithms for sparse polynomials." Symbolic and algebraic computation (1979): 216-226.

[4] de Kleine, Jennifer, Michael Monagan, and Allan Wittkopf. "Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm." Proceedings of the 2005 international symposium on Symbolic and algebraic computation. ACM, 2005.

[5] Go, Soo. Sparse polynomial interpolation and the fast Euclidean algorithm. Diss. Science: Department of Mathematics, 2012.

[6] Javadi, Seyed Mohammad Mahdi. Efficient algorithms for computations with sparse polynomials. Diss. Applied Science: School of Computing Science, 2011.

[7] Ben-Or, Michael, and Prasoon Tiwari. "A deterministic algorithm for sparse multivariate polynomial interpolation." Proceedings of the twentieth annual ACM symposium on Theory of computing. ACM, 1988.

[8] Kaltofen, Erich, Wen-shin Lee, and Austin A. Lobo. "Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm." Proceedings of the 2000 international symposium on Symbolic and algebraic computation. ACM, 2000.

[9] Kaltofen, Erich, and Lakshman Yagati. "Improved sparse multivariate polynomial interpolation algorithms." International Symposium on Symbolic and Algebraic Computation. Springer, Berlin, Heidelberg, 1988.

[10] Hu, Jiaxiong, and Michael Monagan. "A Fast Parallel Sparse Polynomial GCD Algorithm." Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation. ACM, 2016.

[11] Brown, W. Steven. "On Euclid's algorithm and the computation of polynomial greatest common divisors." Journal of the ACM (JACM) 18.4 (1971): 478-504.

[12] Moses, Joel, and David YY Yun. "The ez gcd algorithm." Proceedings of the ACM annual conference. ACM, 1973.

[13] Wang, Paul S. "The eez-gcd algorithm." ACM SIGSAM Bulletin 14.2 (1980): 50-60.

[14] Javadi, Seyed Mohammad Mahdi, and Michael Monagan. "Parallel sparse polynomial interpolation over finite fields." Proceedings of the 4th International Workshop on Parallel and Symbolic Computation. ACM, 2010.