A PROBABILISTIC ALGORITHM FOR COMPUTATION OF POLYNOMIAL GREATEST COMMON WITH SMALLER FACTORS

Yang Zhang[1,2], Xin Qian[1, 2], Qidi You[1,2], Xuan Zhou[1,2],
Xiyong Zhang[1, 2] and Yang Wang[1, 2]

[1]Space star technology co., LTD
[2]State Key Laboratory of Space-Ground Integrated Information Technology

ABSTRACT

In the earlier work, Knuth present an algorithm to decrease the coefficient growth in the Euclidean algorithm of polynomials called subresultant algorithm. However, the output polynomials may have a small factor which can be removed. Then later, Brown of Bell Telephone Laboratories showed the subresultant in another way by adding a variant called τ and gave a way to compute the variant. Nevertheless, the way failed to determine every τ correctly.

In this paper, we will give a probabilistic algorithm to determine the variant τ correctly in most cases by adding a few steps instead of computing $t(x)$ when given $f(x)$ and $g(x) \in \mathbb{Z}[x]$, where $t(x)$ satisfies that $s(x)f(x) + t(x)g(x) = r(x)$, here $t(x), s(x) \in \mathbb{Z}[x]$

KEYWORDS

Euclidean Algorithm, Subresultant, Primitive Remainder Sequences,

1. INTRODUCTION

The Euclidean algorithm and the extended Euclidean algorithm of polynomials is an important research object in polynomial computer algebra. Using this algorithm, one can get the g.c.d. of two polynomials (denoted as $\gcd(f, g)$ when given polynomials $f(x)$ and $g(x)$) and decides whether these polynomials are coprime or not. Specifically, if the degree of $\gcd(f, g)$ is larger than 0, $f(x)$ and $g(x)$ are not coprime, otherwise, $f(x)$ and $g(x)$ are coprime. Being coprime between two polynomials means there exist common roots between these two polynomials.

To quantify the indicator whether there exists a common root between $f(x)$ and $g(x)$, Sylvester gave a matrix in 1840 called Sylvester matrix with entries simply being the coefficients of $f(x)$ and $g(x)$. The determinant of Sylvester matrix is called resultant. Whether the resultant of $f(x)$ and $g(x)$ is nonzero corresponds to the case where $f(x)$ and $g(x)$ are coprime or not respectively. Moreover, Sylvester generalized his definition and introduced the concept of subresultant. They are nonzero if and only if the corresponding degree appears as a degree of a remainder of the Euclidean algorithm.

However, the early Euclidean algorithm of polynomials works for polynomials in $\mathbb{F}[x]$, here $\mathbb{F}$ is a field. In 1836 Jacobi introduced pseudo-division over polynomials and extended the Euclidean

algorithm of polynomials in field to a domain by multiplying $f(x)$ with a certain power of the leading coefficient of $g(x)$ before starting the division.

Using pseudo-division, there are a lot of results about polynomials even the ideal lattice used in cryptography. From 1960, researchers built early computer algebra systems and G.C.D. computations were an important test problem. Nevertheless, using pseudo-division in Euclidean algorithm causes exponential coefficients growth. In 1967, Collins [1] explained that the i-th intermediate coefficients are approximately longer by a factor of $\left(1+\sqrt{2}\right)^{i}$ than the input coefficients. There are many ways to decrease the size of coefficients, most of them are quite inefficient, however. In this paper, we mainly focus on the subresultant algorithm and its variant. In [2], Knuth present the early subresultant algorithm and gave an elegant proof of its correctness. In [3], Brown showed the variant of subresultant algorithm and gave a way to remove the small factor of each remainder. However, the method he present didn't work always.

Recently, in [4], they show an algorithm to triangularize the basis of an ideal lattice which is often used to construct ideal lattice-based cryptosystems. In their algorithm, they need to compute all the PPRSoL (the definition given in Sec.2.4) of $f(x)$ and $g(x)$. However, to obtain the PPRSoL, we need to compute each content of $t_i(x)$ satisfying $s_i(x)f(x)+t_i(x)g(x)=r_i(x)$ to remove the extra factor in each original remaindes $r_i(x)$. However, in this paper, we find a new way to obtain PPRSoL without computing $t_i(x)$ by applying the variant of subresultant algorithm.

In this paper, we give some results about the extended Euclidean algorithm. Using these results, we propose a new algorithm that outputting the PPRSoL of $f(x)$ and $g(x)$ which works for most cases.

2. PRELIMINARIES

2.1. Notations

In this paper, a matrix is denoted as uppercase bold letter and a vector is denoted as lowercase bold letter. For a matrix$\boldsymbol{A}\in\mathbb{R}^{m\times n}$, the element in the i-th row and the j-th column of $\boldsymbol{A}$ is expressed as $a_{i,j}$. For a polynomial $f(x)$ with degree n, we use $lc(f)$ to present the leading coefficient of $f(x)$ and use$deg(f)$ to present the degree. The degree of a constant polynomial is defined as 0 and the degree of a zero polynomial is defined as $-\infty$. The greatest common divisor is abbreviated to g.c.d.. Let $\varsigma[x]$ denote the domain of polynomials in x with coefficients in ς. Unless otherwise specified, we only consider the polynomials in $\mathbb{Z}[x]$.

2.2. Some Definitions

Definition 1: [Hermite Normal Form] Given a square matrix $\boldsymbol{H}\in\mathbb{Z}^{n\times n}$. Then $\boldsymbol{H}$ is Hermite Normal Form(HNF) if and only if it satisfies:

1) $h_{i,i}\geq 1$, for$1\leq i\leq n$;
2) $h_{i,j}=0$, for $1\leq j\leq i\leq n$;
3) $h_{j,i}<h_{i,i}$, for $1\leq j<i\leq n$.

We need to emphasize that the definition given above is only one way to define HNF. According to row or column transformation and upper or lower triangularization, there are other different definitions of HNF.

Definition 2: [Primitive Polynomial] A polynomial $a(x) \in \mathbb{Z}[x]$ is called a primitive polynomial if for any integer $|k| > 1$, $a(x)/k \notin \mathbb{Z}[x]$.

Definition 3: [Content] For a polynomial $a(x) = a_n x^n + \cdots + a_1 x + a_0 \in \mathbb{Z}[x]$, the content of $a(x)$, denoted as$cont\big(a(x)\big)$, is the g.c.d of $(a_n, \cdots, a_1, a_0)$.

Definition 3: [Resultant] Let $f(x) = f_n x^n + \cdots + f_1 x + f_0$, $g(x) = g_m x^m + \cdots + g_1 x + g_0$ be two polynomials with degree n and m respectively. Define the Sylvester matrix of $f(x)$ and $g(x)$ as

$$\mathbf{Sylv}(f,g) = \begin{bmatrix} x^{m-1}f(x) \\ x^{m-2}f(x) \\ \vdots \\ f(x) \\ x^{n-1}g(x) \\ x^{n-2}g(x) \\ \vdots \\ g(x) \end{bmatrix} = \begin{bmatrix} f_n & f_{n-1} & \cdots & f_0 & & & \\ & f_n & f_{n-1} & \cdots & f_0 & & \\ & & \ddots & & & \ddots & \\ & & & f_n & f_{n-1} & \cdots & f_0 \\ g_m & g_{m-1} & \cdots & g_1 & g_0 & & \\ & g_m & g_{m-1} & \cdots & g_1 & g_0 & \\ & & \ddots & & & \ddots & \\ & & & g_m & g_{m-1} & \cdots & g_1 & g_0 \end{bmatrix}_{(m+n)\times(m+n)}$$

Then the resultant of $f(x)$ and $g(x)$, denoted as $\mathrm{Res}\big(f(x), g(x)\big)$, is the determinant of $\mathbf{Sylv}(f,g)$.

Definition 3: [Subresultant] Let $f(x) = f_n x^n + \cdots + f_1 x + f_0$, $g(x) = g_m x^m + \cdots + g_1 x + g_0$ be two polynomials with degree n and m respectively.For $0 \leq k < n$, the k-th subresultant of $f(x)$ and $g(x)$ is the determinant of $S_k(f,g)$ defines as

$$S_k(f,g) = \begin{bmatrix} f_n & f_{n-1} & \cdots & f_{n-m+k+1} & \cdots & f_{k+1} & \cdots & f_{2k-m+1} \\ & f_n & \cdots & f_{n-m+k+2} & \cdots & f_{k+2} & \cdots & f_{2k-m+2} \\ & & \ddots & \vdots & & \vdots & & \vdots \\ & & & f_n & \cdots & f_m & \cdots & f_k \\ g_m & g_{m-1} & \cdots & g_{k+1} & \cdots & g_{m-n+k+1} & \cdots & g_{2k-n+1} \\ & g_m & & \vdots & & \vdots & & \vdots \\ & & \ddots & \vdots & & \vdots & & \vdots \\ & & & g_m & & \vdots & & \vdots \\ & & & & \ddots & \vdots & & \vdots \\ & & & & & g_m & \cdots & g_k \end{bmatrix}_{(m+n-2k)\times(m+n-2k)}$$

Remark 1. From the structure of $S_k(f,g)$ and $\mathbf{Sylv}(f,g)$, we can tell that indeed if we delete the last $2k$ columns and the last k rows of $f(x)$ and $g(x)$ respectively in $\mathbf{Sylv}(f,g)$, we obtain $S_k(f,g)$. Expecially, $S_0(f,g) = \mathbf{Sylv}(f,g)$.

Next we will give the conception of ideal lattice which takes an important role in the lattice-based cryptography, and we mainly focus on the cases in which an ideal lattice can be derived from $f(x)$ and $g(x)$.

Ideal Lattice We define ideal lattice over a ring $R = \mathbb{Z}[x]/\langle f(x)\rangle$, where $f(x) \in \mathbb{Z}[x]$ is a monic and irreducible polynomial of degree n and $\langle f(x)\rangle$ is the ideal generated by $f(x) \in \mathbb{Z}[x]$.

Consider the coefficient embedding

$$\begin{array}{ccc} \sigma : R & \mapsto & \mathbb{Z}^n \\ \sum_{i=0}^{n-1} a_i x^i & \mapsto & (a_{n-1}, a_{n-2}, \cdots, a_0) \end{array}$$

From [5], we know that the ideal generated by $g(x)$ forms a lattice under σ and we call it the ideal lattice $\mathcal{L}$ generated by $g(x)$. Moreover, $g(x)\mathrm{mod}f(x)$, $xg(x)\mathrm{mod}f(x)$, $\cdots$, $x^{n-1}g(x)\mathrm{mod}f(x)$ form a basis of $\mathcal{L}$. As we can see, the basis is closely related to the Sylvester matrix of $f(x)$ and $g(x)$. When $f(x)$ and $g(x)$ are coprime over $\mathbb{Q}[x]$, the ideal lattice is full-rank.

Then we present a lemma in [5] that we will use later.

Lemma 1. Let $\mathcal{L}$ be the ideal lattice generated by $g(x) \in R = \mathbb{Z}[x]/\langle f(x)\rangle$, where $f(x)$ is a monic polynomial of degree n and is relatively prime to $g(x)$. Then the Hermite Normal Form of a basis of $\mathcal{L}$

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ & h_{2,2} & \cdots & h_{2,n} \\ & & \ddots & \vdots \\ & & & h_{n,n} \end{bmatrix}$$

satisfies $h_{i,i} | h_{l,j}$, for $1 \le i \le l \le j \le n$.

2.3. The Classical Euclidean Algorithm of Polynomials over A Field

Given a field $\mathbb{F}$. Let $f(x)$ and $g(x) \in \mathbb{F}[x]$ with $deg(f) > deg(g)$. Then the division of $f(x)$ and $g(x)$ yields a unique quotient $Q(x)$ and remainder $R(x)$ such that

$$f(x) = Q(x)g(x) + R(x)$$

here $deg(g) > deg(r)$, $deg(q) = deg(f) - deg(g)$.

If we repeat the step for each divisor polynomial and remainder, we will obtain a sequence of remainders with decreasing degrees. Formally, a detailed procedure of the Euclidean algorithm of polynomials over a field is present as following:

$$\begin{cases} f(x) &= Q_1(x)g(x) + R_1(x) \\ g(x) &= Q_2(x)R_1(x) + R_2(x) \\ & \vdots \\ R_{l-2}(x) &= Q_l(x)R_{l-1}(x) + R_l(x) \\ R_{l-1}(x) &= Q_{l+1}(x)R_l(x) \end{cases}$$

where$deg(g) > deg(R_1) > \cdots > deg(R_l)$ and all the coefficients are in the given field. Note that if $\deg(R_l) = 0$, it shows that $f(x)$ and $g(x)$ are coprime in $\mathbb{F}[x]$, which means the resultant of $f(x)$ and $g(x)$ is nonzero.

2.4. Polynomial Remainder Sequence

The procedure of the Euclidean algorithm of polynomials over a unique factorization domain (UFD) is similar to the one over a field. The difference exits because the division between two polynomials requires exact divisibility in the given domain, which is usually impossible to realize. To solve the problem, the procedure of pseudo-division is proposed, which yields a unique pseudo-quotient $q(x)$ and pseudo-remainder $r(x)$ such that

$$\big(\mathrm{lc}(g)\big)^{\delta+1} f(x) = q(x)g(x) + r(x)$$

here $deg(g) > deg(r)$, $\delta = deg(f) - deg(g)$, $r(x)$ is equivalent with $\mathrm{prem}(f, g)$. Moreover, the coefficients of $q(x)$ and $r(x)$ are in the given domain.

For nonzero polynomials $a(x), b(x) \in \varsigma[x]$, we say $a(x)$ is similar to $b(x)(a(x) \sim b(x))$ if there exist $c_1, c_2 \in \varsigma$ such that $c_1 a(x) = c_2 b(x)$. So if we choose $r'(x)$ that is similar to $r(x)$, we can do the same step as above for $g(x)$ and $r'(x)$. Thus, we can rewrite the procedure of pseudo-division:

$$\alpha f(x) = q(x)g(x) + \beta r(x).$$

Then the detailed procedure of pseudo-division is present as following:

$$\begin{cases} \alpha_1 f(x) &= q_1(x)g(x) + \beta_1 r_1(x) \\ \alpha_2 g(x) &= q_2(x)r_1(x) + \beta_2 r_2(x) \\ & \vdots \\ \alpha_l r_{l-2}(x) &= q_l(x)r_{l-1}(x) + \beta_l r_l(x) \\ \alpha_{l+1} r_{l-1}(x) &= q_{l+1}(x)r_l(x) \end{cases}$$

here $deg(g) > deg(r_1) > \cdots > deg(r_l)$ and all theα_i and β_i are in the given domain.

Generally, we denote $f(x) = r_{-1}(x)$ and $g(x) = r_0(x)$, then $\alpha_i = \big(lc(\mathrm{r}_{i-1})\big)^{\delta_{i-2}-1}$, where $\delta_i = deg(r_i) - deg(r_{i+1})$. Note that now $\mathrm{prem}(r_{i-2}, r_{i-1}) = \beta_i r_i(x)$. Then $r_{-1}(x)$, $r_0(x)$,$\cdots$,$r_l(x)$ form a sequence called polynomial remainder sequence(PRS).

From [5], if a remainder $r(x) = s(x)f(x) + t(x)g(x)$ can derive a basis of ideal lattice, $t(x)$ must be primitive. In this paper, we also want to obtain such remainders and we call these remainders as primitive PRS of lattice (PPRSoL). Next, we give a result about $s_i(x)$ and $t_i(x)$.

Lemma 2. Let $f(x)$, $g(x) \in \mathbb{Z}[x]$ be two polynomials with degree n and m respectively, where $n > m$. Let $r_{-1}(x), r_0(x), \cdots, r_l(x)$ be the remainders in procedure of pseudo-division. If $deg(r_i) = n_i$, then $r_i(x) = s_i(x)f(x) + t_i(x)g(x)$ satisfies $deg(s_i) < m$, $deg(t_i) < n$ and:

1) $s_i(x) = \big(\alpha_i s_{i-2}(x) - q_i(x)s_{i-1}(x)\big)/\beta_i, t_i(x) = \big(\alpha_i t_{i-2}(x) - q_i(x)t_{i-1}(x)\big)/\beta_i$
2) $deg(s_i) = m - deg(r_{i-1}), deg(t_i) = n - deg(r_{i-1})$

If we represent $r_i(x) = s_i(x)f(x) + t_i(x)g(x)$ under the embedding σ, for $i = -1, 0, \cdots, l$, then we can denote $r_i(x)$ as a sequence of vectors and we use a matrix $\boldsymbol{R}$ to represent $r_i(x)$ as following:

$$\begin{bmatrix}
f_n & \cdots & f_{n-m+1} & f_{n-m} & \cdots & f_0 & & & & & & & & \\
 & \ddots & & & & & & & \ddots & & & & & \\
 & & f_n & f_{n-1} & \cdots & f_{m-1} & f_{m-2} & & \cdots & & & f_1 & f_0 \\
 & & & r_{0,n_0} & \cdots & r_{0,0} & & & & & & & \\
 & & & & \ddots & & & & & & & & \\
\vdots & & & & & r_{0,n_0} & \cdots & r_{0,0} & & & & & \\
 & & & & & & r_{1,n_0} & \cdots & r_{1,0} & & & & \\
\mathbf{0} & & & & & & & \ddots & & & & & \\
 & & & & & & & & r_{1,n_1} & \cdots & r_{1,0} & & \\
 & & & & & & & & & \ddots & & & \\
\vdots & & & & & & & & & & r_{l,n_l} & & \\
 & & & & & & & & & & & \ddots & \\
 & & & & & & & & & & & & r_{l,n_l}
\end{bmatrix}_{(m+n)\times(m+n)}$$

Also, we use $\boldsymbol{S}$ and $\boldsymbol{T}$ to represent the matrice denoting $s_i(x)$ and $t_i(x)$ respectively.

$$\boldsymbol{T} = \begin{bmatrix}
 & & & & & & 1 & & \\
 & & & & & & & \ddots & \\
 & & & & & & & & 1 \\
 & & t_{1,n-n_0} & & \cdots & & t_{1,0} & & \\
 & & & \ddots & & & & \ddots & \\
 & & & & t_{1,n-n_0} & & \cdots & & t_{1,0} \\
 & & & \ddots & & & & & \\
t_{l,n-n_{l-1}} & & & & \cdots & & t_{l,0} & & \\
 & \ddots & & & & & & \ddots & \\
 & & t_{l,n-n_{l-1}} & & & \cdots & & & t_{l,0}
\end{bmatrix}_{n\times n}$$

$$\boldsymbol{S}=\begin{bmatrix} & & & & & & 0 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 0 \\ & & s_{1,m-n_0} & & \cdots & & s_{1,0} & & \\ & & & \ddots & & & & \ddots & \\ & & & & s_{1,m-n_0} & & \cdots & & s_{1,0} \\ & & & \ddots & & & & & \\ s_{l,m-n_{l-1}} & & & & \cdots & & s_{l,0} & & \\ & \ddots & & & & & & \ddots & \\ & & s_{l,m-n_{l-1}} & & & \cdots & & & s_{l,0} \end{bmatrix}_{n\times m}$$

So if we give a matrix named $\boldsymbol{ST}$, then procedure of pseudo-division can be represent as a matrix multiplication.

$$\boldsymbol{ST}=\begin{bmatrix} 1 & & & \\ & \ddots & & \boldsymbol{0} \\ & & 1 & \\ & \boldsymbol{S} & & \boldsymbol{T} \end{bmatrix}_{(m+n)\times(m+n)}$$

$$\boldsymbol{ST}\cdot\mathbf{Sylv}(f,g)=\boldsymbol{R}$$

Here we need to show that by elementary row transformation, $\boldsymbol{ST}$ can be transformed into

$$\boldsymbol{ST}=\begin{bmatrix} 1 & & & \\ & \ddots & & \boldsymbol{0} \\ & & 1 & \\ & \boldsymbol{0} & & \boldsymbol{T} \end{bmatrix}_{(m+n)\times(m+n)}$$

which means that the determinant of $\boldsymbol{ST}$ equals to the determinant of $\boldsymbol{T}$. Also according to lemma 2, it turns out that after appropriate row switching, $\boldsymbol{T}$ is actually an upper triangular matrix, thus the determinant of $\boldsymbol{T}$ is $\left|\prod_{i=0}^{l}\mathrm{lc}(t_i)^{n_{i-1}-n_i}\right|$.

In the following part, we introduce some typical PRSs which differs from each other by choosing different β_i.

2.4.1. Euclidean Polynomial Remainder Sequences

When choosing $\beta_i=1$ for all i in PRS, we obtain Euclidean PRS. This is a generalization of the Euclidean algorithm over integers. However, the algorithm is quite inefficient because with the proceeding of the sequence, the coefficients of the remainders grow exponentially. To be specific, we need to calculate each $t_i(x)$ and $\mathrm{cont}\big(t_i(x)\big)$ to get a eligible PPRSoL, which costs too much. So we need to determine certain β_i to ensure the efficiency.

2.4.2. Primitive Polynomial Remainder Sequences

When choosing $\beta_i = \mathrm{cont}\big(\mathrm{prem}(r_{i-2}, r_{i-1})\big)$ for all i in PRS, we obtain primitive PRS. Although the algorithm stops the coefficients growing exponentially in every step of the pseudo-division, however, when proceeding primitive PRS, the coefficients of $s_i(x)$ and $t_i(x)$ may be not in the given domain, which means that the PRS we obtain is not PPRSoL. So primitive PRS doesn't satisfy our requirement.

2.4.3. Subresultant Polynomial Remainder Sequences

When β_i is related to the subresultant, we obtain subresultant PRS. The equation set as following depicts the procedure of the subresultant PRS algorithm in [2].

$$\begin{cases} \alpha'_1 f(x) = q'_1(x)g(x) + \beta'_1 r'_1(x) \\ \alpha'_2 g(x) = q'_2(x)r_1(x) + \beta'_2 r'_2(x) \\ \vdots \\ \alpha'_l r'_{l-2}(x) = q'_l(x)r_{l-1}(x) + \beta'_l r'_l(x) \end{cases}$$

where $r_{-1}(x) = f(x)$, $r_0(x) = g(x)$, $n_i = deg(r'_i)$, $\delta_i = n_i - n_{i+1}$, $\alpha'_i = \big(\mathrm{lc}(r'_{i-1})\big)^{\delta_{i-2}+1}$, $\beta'_i = \mathrm{lc}(r'_{i-2}){h_i}^{\delta_{i-2}}$, $h_1 = 1$, $h_i = (lc(r'_{i-2}))^{\delta_{i-3}}{h_{i-1}}^{1-\delta_{i-3}}$, for $2 \le i \le l+1$.

Intuitively, the intact subresultant algorithm can be present in Algorithm 1. We point out that because we want to get PPRSoL, the input of every PRS algorithm in the paper contains a monic and irreducible polynomial.

Algorithm 1 Subresultant PRS Algorithm

Input: two polynomials $f(x), g(x) \in \mathbb{Z}[x]$ with degree n and m respectively and $f(x)$ is monic and irreducible
Output: Subresultant PRS, $r'_0(x), r'_1(x), \cdots$
1.[Initialize] $l \leftarrow h \leftarrow 1, r'_0(x) = g(x), i \leftarrow 1$
2.[Pseudo-division]
2.1 Set $\delta = deg(f) - deg(g)$
2.2 Calculate $r(x)$ such that $r(x) = s(x)f(x) + t(x)g(x)$
3.[Adjust remainder]
3.1 $u(x) \leftarrow g(x), r'_i(x) \leftarrow g(x) \leftarrow r(x)/lh^{\delta}$
3.2 $l \leftarrow lc(f), h \leftarrow h^{1-\delta}l^{\delta}$
3.3 If $deg(r) = 0$, go to Step 4
3.4 $i \leftarrow i + 1$, go to Step 2
4.[Return] $r'_0(x), r'_1(x), \cdots$

Notice that for $r'_i(x) = s'_i(x)f(x) + t'_i(x)g(x)$, $t'_i(x)$ maybe not primitive in the given domain, which means that we can still decrease the coefficients of $r'_i(x)$ by removing a factor.

In [6], the author shows that the h_i is indeed the n_{i-1}-th subresultant of $f(x)$ and $g(x)$, that is $h_i = S_{n_{i-1}}(f, g)$. Also, in [3], the author shows that every h_i is an integer and $r'_i(x) \in \mathbb{Z}[x]$ and gives an elegant proof.

2.4.4. Improvements of Subresultant Polynomial Remainder Sequences

This is another expression of subresultant PRS. As stated above, for the output of Algorithm 1, $r'_i(x) = s'_i(x)f(x) + t'_i(x)g(x)$, $t'_i(x)$ maybe not primitive and there might exist a divisor τ_i such that $t_i(x) = t'_i(x)/\tau_i$ is primitive. So in the improvement version, the author transforms the procedure of the subresultant PRS algorithm as following,

$$\begin{cases} \alpha_1 f(x) = q_1(x)g(x) + \beta_1 r_1(x) \\ \alpha_2 g(x) = q_2(x)r_1(x) + \beta_2 r_2(x) \\ \vdots \\ \alpha_l r_{l-2}(x) = q_l(x)r_{l-1}(x) + \beta_l r_l(x) \end{cases}$$

where $r_{-1}(x) = f(x)$, $r_0(x) = g(x)$, $h_1 = 1$, $n_i = deg(r_i)$, $\delta_i = n_i - n_{i+1}$, $\alpha_i = \left(lc(r_{i-1})\right)^{\delta_{i-2}+1}$, $\beta_i = lc(r_{i-2}){h_i}^{\delta_{i-2}}{\tau_{i-1}}^{-\delta_{i-2}-1}\tau_i$, $h_i = (\tau_{i-2}lc(r_{i-2}))^{\delta_{i-3}}{h_{i-1}}^{1-\delta_{i-3}}$, for $2 \le i \le l+1$. τ_i is an integer such that $t'_i(x)/\tau_i$is a primitive polynomial. Clearly, $\tau_0 = 1$. In [3], the author chose $\tau_i = \mathrm{lc}(r_{i-1})$ if $\mathrm{lc}(r_{i-1})|r'_i(x)$, otherwise $\tau_i = 1$. However, the method to choose τ_i doesn't work for every τ_i.

Comparing the two subresultant algorithms, we need to emphasis that all the h_is are equal in the two algorithms.

3. Some Properties of the Sub Resultant Polynomial Remainder Sequence

Before presenting our algorithm, we give some results about the subresultant PRS.

Proposition 1. Given two polynomials $a(x) = a_n x^n + \cdots + a_1 x + a_0$ and $b(x) = b_m x^m + \cdots + b_1 x + b_0 \in \mathbb{Z}[x]$, where $n > m$. Write $b_m^{n-m+1}a(x) = q(x)b(x) + r(x)$. Define the matrix

$$\boldsymbol{M} = \begin{bmatrix} a_n & a_{n-1} & \cdots & a_{n-m+1} & a_{n-\mathrm{m}} & \cdots & a_2 & a_1 & a_0 \\ b_m & b_{m-1} & \cdots & b_1 & b_0 & & & & \\ & b_m & b_{m-1} & \cdots & b_1 & b_0 & & & \\ & & & \ddots & & & & & \\ & & & & b_{\mathrm{m}} & b_{\mathrm{m\text{-}1}} & \cdots & b_1 & b_0 \end{bmatrix}$$

If the determinant of the matrix $\boldsymbol{M}_i$ is denoted as Δ_i, where $\boldsymbol{M}_i$ is the $i \times i$ submatrix of $\boldsymbol{M}$ obtained by deleting the last $(n-m+2-i)$ rows and the last $(n+1-i)$ columns from $\boldsymbol{M}$, $i = 0, \ldots, n-m+1$. Then $q(x) = \sum_{i=0}^{n-m} \Delta_{n-m+1-i} b_m^i x^i$. Moreover, we have $\left(cont\big(a(x)\big)cont\big(b(x)\big)^{n-m}\right)|q(x)$.

Proof. We first give the detail of the pseudo-division procedure,

$$\begin{cases} b_m a(x) = a_n x^{n-m} b(x) + R_1(x) \\ b_m R_1(x) = lc(R_1)x^{n-m-1}b(x) + R_2(x) \\ \vdots \\ b_m R_{n-m-1}(x) = lc(R_{n-m-1})xb(x) + R_{n-m}(x) \\ b_m R_{n-m}(x) = lc(R_{n-m})xb(x) + r(x) \end{cases}$$

We denote $R_0(x) = a(x)$ and $R_{n-m+1}(x) = r(x)$, then we claim that $R_i(x) = \sum_{j=0}^{n-i} \bar{\Delta}_{i,j} x^j$, where $\bar{\Delta}_{i,j}$ is the determinant of the $(i+1) \times (i+1)$ matrix $M_{i,j}$ obtained by deleting the last $(n-m+1-i)$ rows and the last $(n+1-i)$ columns except column $(n+1-j)$ from M, $i = 0, \dots, n-m+1, j = 0, \dots, n-i$. Clearly, $\Delta_{i+1} = \bar{\Delta}_{i,n-i}$.

Then we explain the claim by induction on i, $i = 0, \dots, n-m+1$.

For $i = 0$, we have $R_0(x) = a(x)$ and it's obvious that $a_j = \bar{\Delta}_{0,j}$ for $j = 0, \dots, n$.

Next we assume that the claim holds for $i = k-1$. Then we denote $b_m R_{k-1}(x)$ and $b(x)$ as following,

$$\begin{bmatrix} b_m\bar{\Delta}_{k-1,n+1-k} & b_m\bar{\Delta}_{k-1,n-k} & \cdots & b_m\bar{\Delta}_{k-1,n-m+1-k} & \cdots & \cdots & b_m\bar{\Delta}_{k-1,1} & b_m\bar{\Delta}_{k-1,0} \\ b_m & b_{m-1} & \cdots & b_0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Then the coefficient of $x^{n-k+1-i}$ in $R_k(x)$ is $b_m\bar{\Delta}_{k-1,n+1-k-i} - b_{m-i}\bar{\Delta}_{k-1,n+1-k}$ if $1 \le i \le m$ and $b_m\bar{\Delta}_{k-1,n+1-k-i}$ otherwise. According to the structure of M we know that the coefficient of $x^{n-k+1-i}$ is exactly $\bar{\Delta}_{k,n-k+1-i}$. So the claim holds.

From the claim we have $lc(R_i) = \bar{\Delta}_{i,n-i} = \Delta_{i+1}$, so $q(x) = \sum_{i=0}^{n-m} lc(R_{n-m-i}) b_m^i x^i = \sum_{i=0}^{n-m} \Delta_{n-m+1-i} b_m^i x^i$.

Then from the structure of M_i, we know $\left(cont(a(x))cont(b(x))^{n-m-i}\right) | \Delta_{n-m+1-i}$. So $\left(cont(a(x))cont(b(x))^{n-m}\right) | \Delta_{n-m+1-i} b_m^i$, which means $\left(cont(a(x))cont(b(x))^{n-m}\right) | q(x)$.

Proposition 2. Let $r_1(x), \cdots, r_l(x)$ be the remainders obtained in improved subresultant algorithm. Present $r_i(x) = s_i(x)f(x) + t_i(x)g(x)$, for $i = 1, \cdots, l$. Then we have $lc(t_i) = \frac{h_{i+1}}{\tau_i}$.

Proof. According to Lemma 2, $t_i(x) = \frac{1}{\beta_i}(\alpha_i t_{i-2}(x) - q_i(x) t_{i-1}(x))$ and $deg(t_i) = n - n_{i-1}$. Also $deg(q_i) = \delta_{i-2}$, so $deg(t_i) = n - n_{i-3} < deg(q_i t_{i-1}) = n - n_{i-1}$. Then $lc(t_i) = \frac{1}{\beta_i} lc(q_i)\, lc(t_{i-1})$, so $lc(q_i) = \frac{lc(r_{i-2})}{lc(r_{i-1})}\alpha_i$. Then $lc(t_i) = \frac{lc(r_{i-2})\alpha_i}{lc(r_{i-1})\beta_i} lc(t_{i-1}) = \frac{\alpha_1 \dots \alpha_i}{\beta_1 \dots \beta_i lc(r_{i-1})}$.

Because $\alpha_i = \left(lc(r_{i-1})\right)^{\delta_{i-2}+1}$, $\beta_i = lc(r_{i-1}) {h_i}^{\delta_{i-2}} {\tau_{i-1}}^{-\delta_{i-2}-1} \tau_i$, then we have

$$\begin{aligned} lc(t_i) &= \frac{1}{lc(r_{i-1})} \frac{(\tau_{i-1}\, lc(r_{i-1}))^{\delta_{i-2}+1}}{lc(r_{i-2})\, {h_i}^{\delta_{i-2}} \tau_i} \frac{(\tau_{i-2}\, lc(r_{i-2}))^{\delta_{i-3}+1}}{lc(r_{i-3})\, {h_{i-1}}^{\delta_{i-3}} \tau_{i-1}} \dots \frac{(\tau_0\, lc(r_0))^{\delta_{-1}+1}}{lc(r_{-1})\, {h_1}^{\delta_{-1}} \tau_1} \\ &= \frac{1}{\tau_i} \frac{(\tau_{i-1}\, lc(r_{i-1}))^{\delta_{i-2}}}{{h_i}^{\delta_{i-2}}} \frac{(\tau_{i-2}\, lc(r_{i-2}))^{\delta_{i-3}}}{{h_{i-1}}^{\delta_{i-3}}} \dots \frac{(\tau_0\, lc(r_0))^{\delta_{-1}}}{{h_1}^{\delta_{-1}}} \\ &= \frac{1}{\tau_i} \frac{(\tau_{i-1}\, lc(r_{i-1}))^{\delta_{i-2}}}{{h_i}^{\delta_{i-2}}} \frac{(\tau_{i-2}\, lc(r_{i-2}))^{\delta_{i-3}}}{{h_{i-1}}^{\delta_{i-3}}} \dots \frac{(\tau_0\, lc(r_1))^{\delta_0}}{{h_1}^{\delta_{-1}}} h_2 \\ &= \dots = \frac{h_{i+1}}{\tau_i} \end{aligned}$$

Remark 2. If we do similar steps for ${r'}_0(x), {r'}_1(x), \cdots, {r'}_l(x)$ in Algorithm 1 and present each ${r'}_i(x) = {s'}_i(x)f(x) + {t'}_i(x)g(x)$, then we obtain $lc({t'}_i) = h_{i+1}$.

Before giving next lemmas, we first present a useful algorithm from [5]. We use the same symbols in [5], $\{n - n_{i-1} + 1, \cdots, n - n_i\} = I_i$, then $\{1,2, \cdots, n\} = \cup_{i+1}^{l} I_i$.

Algorithm 2 A Useful Algorithm

Input:$r_0(x), r_1(x), \cdots, r_l(x)$from improved subresultant algorithm
Output: $\bar{r}_0(x), \bar{r}_1(x), \cdots, \bar{r}_l(x)$
1.When$k \in I_l$, $r'_k(x) = r_l(x)x^{n-k}$,$i \leftarrow\ l-1$
2.When $k \in I_i$
2.1 Set Compute ϕ and ψ, such that $\psi lc(r_i) + \phi lc(\bar{r}_{i+1}) = \gcd\big(lc(r_i), lc(\bar{r}_{i+1})\big)$
2.2 Set $\bar{r}_i(x) = \psi r_i(x) + \phi \bar{r}_{i+1}(x)x^{\delta_i}$
2.3 If $lc\big(\bar{r}_{n-n_i}\big) = 1$, set $\bar{r}_j(x) = \bar{r}_{n-n_i}(x)x^{n-n_i-j}, j = 1, \cdots, n-n_i$, go to Step 3; otherwise $\bar{r}_k(x) = \bar{r}_{n-n_i}(x)x^{n-n_i-k}$, $i \leftarrow\ l-1$
2.4If $i >\ 0$, go to Step 2, otherwise go to Step 3
4.Return$\bar{r}_0(x), \bar{r}_1(x), \cdots, \bar{r}_l(x)$

We need to explain that Algorithm 2 is equivalent to the corresponding algorithm in [4] and we just use polynomials to express the output instead of a matrix in [4].

Then we will present some results of $cont\big(r_i(x)\big)$and $lc(\bar{r}_i)$.

Lemma 3. Let $r_1(x), \cdots, r_l(x)$ be the polynomial remainder sequence obtained in improved subresultant algorithm. Then $cont\big(r_i(x)\big)|cont\big(r_{i-1}(x)\big)$ for $0 \le i \le l-1$.

Proof. We prove this lemma by induction on i, $i = 0, \ldots, l-1$.

Suppose that H is the Hermite Normal Form over the ideal lattice $\mathcal{L}$ generated by $g(x) \in \mathbb{Z}[x]/\langle f(x)\rangle$, and $r_i(x)$ belongs to $\mathcal{L}$. When $i = 0$, because $r_0(x)$ generates the ideal lattice $\mathcal{L}$, then all the vectors in $\mathcal{L}$ can be divided exactly by $cont(r_0(x))$.

Next we suppose that when $i \le k-1$, $cont\big(r_{i-1}(x)\big)|cont\big(r_i(x)\big)$, then we need to show that $cont\big(r_k(x)\big)|cont\big(r_{k+1}(x)\big)$.

Consider the $(k+1)$-th equation in improved subresultant algorithm,

$$\alpha_{k+1}r_{k-1}(x) = q_{k+1}(x)r_k(x) + \beta_{k+1}(x)r_{k+1}(x),$$

then we know $cont\big(r_{k-1}(x)\big)cont\big(r_k(x)\big)^{\delta_{k-1}+1}|\beta_{k+1}r_{k+1}(x)$. Because

$$t_{k+1}(x) = (\alpha_{k+1}t_{k-1}(x) - q_{k+1}(x)t_k(x))/\beta_{k+1},$$

β_{k+1} must contain a factor as the content of $\alpha_{k+1}t_{k-1}(x) - q_{k+1}(x)t_k(x)$. Also $\alpha_{k+1} = lc(r_k)^{\delta_{k-1}+1}$, $(cont\big(r_{k-1}(x)\big)cont\big(r_k(x)\big)^{\delta_{k-1}})|q_{k+1}(x)$ due to Proposition 1. Based on the assumption$(cont\big(r_{k-1}(x)\big)|cont\big(r_k(x)\big)$, so $\Big(cont\big(r_{k-1}(x)\big)cont\big(r_k(x)\big)^{\delta_{k-1}}\Big)|\beta_{k+1}$.

If $(cont\big(r_k(x)\big) \nmid cont\big(r_{k+1}(x)\big)$, then there exists a prime a such that $a|cont\big(r_k(x)\big)$ and $\Big(a \cdot cont\big(r_{k-1}(x)\big)cont\big(r_k(x)\big)^{\delta_{k-1}}\Big)|\beta_{k+1}$. We give 2 cases as following:

1) $a \cdot cont\big(r_{k-1}(x)\big) \nmid cont\big(r_k(x)\big)$, which means that $a \nmid cont\big(r_{k-1}(x)\big)$ and $a \nmid \frac{cont(r_k(x))}{cont(r_{k-1}(x))}$. According to Proposition 1, we know $r_{k+1}(x) = \sum_{j=0}^{n_k-1} \bar{\Delta}_{n_k-1,j}\, x^j/\beta_{k+1}$, here

$\bar{\Delta}_{n_k-1,j}$ is the determinant of the $(\delta_{k-1}+2)\times(\delta_{k-1}+2)$ matrix obtained by deleting the last n_k columns except column $n_{k-1}+1-j$ from M, $j=0,\dots,n_k-1$. Because $a\nmid \frac{r_{k-1}(x)}{cont(r_{k-1}(x))}$, there exits a $j>1$ such that $a|M_j(x)$, which means $a|\frac{lc(r_k)}{cont(r_k(x))}$. Thus we obtan $\left(a\cdot cont(r_{k-1}(x))cont(r_k(x))^{\delta_{k-1}}\right)|\alpha_{k+1}$. According to equation $t_{k+1}(x)=(\alpha_{k+1}t_{k-1}(x)-q_{k+1}(x)t_k(x))/\beta_{k+1}$,we have that

$$\left(a\cdot cont(r_{k-1}(x))cont(r_k(x))^{\delta_{k-1}}\right)|q_{k+1}(x).$$

$\left(a\cdot cont(r_{k-1}(x))cont(r_k(x))^{\delta_{k-1}}\right)|\beta_{k+1}r_{k+1}(x)$, which means we have get $cont(r_k(x))|cont(r_{k+1}(x))$.

2) $a\cdot cont(r_{k-1}(x))|cont(r_k(x))$. Because $\alpha_{k+1}=lc(r_k)^{\delta_{k-1}+1}$, then we have result that $cont(r_k(x))^{\delta_{k-1}+1}|\alpha_{k+1}$, thus $\left(a\cdot cont(r_{k-1}(x))cont(r_k(x))^{\delta_{k-1}}\right)|\alpha_{k+1}$. As the same step in case 1, we still get $cont(r_k(x))|cont(r_{k+1}(x))$.

So in conclusion we obtain $cont(r_k(x))|cont(r_{k+1}(x))$. The proof is completed.

Lemma 4. Let $r_1(x),\cdots,r_l(x)$ be the polynomial remainder sequence obtained in improved resultant algorithm and $\bar{r}_1(x),\cdots,\bar{r}_l(x)$ be the output of Algorithm 2. If $\gcd(lc(r_i),cont(r_{i+1}(x)))=cont(r_i(x))$ for $i\le l-1$, then $lc(\bar{r}_i)=cont(r_i(x))$. Moreover, $lc(\bar{r}_i)|\bar{r}_l(x)$.

Proof. We notice that from Algorithm 2, $\bar{r}_i(x)=\psi r_i(x)+\phi\bar{r}_{i+1}(x)x^{\delta_i}$, where ψ and ϕ satisfy $\psi lc(r_i)+\phi lc(\bar{r}_{i+1})=\gcd(lc(r_i),lc(\bar{r}_{i+1}))=lc(\bar{r}_i)$. If we already have $\gcd(lc(r_i),cont(r_{i+1}(x)))=cont(r_i(x))$ and $cont(r_{i+1}(x))=lc(\bar{r}_{i+1})$, then we have $lc(\bar{r}_i)=cont(r_i(x))$.

When $i=l$, this is a trivial result because $lc(\bar{r}_l)=\bar{r}_l(x)=cont(r_l(x))$. So we know $lc(\bar{r}_{l-1})=cont(r_{l-1}(x)),lc(\bar{r}_{l-2})=cont(r_{l-2}(x)),\dots,$ and so on. Thus, if $\gcd(lc(r_i),cont(r_{i+1}(x)))=cont(r_i(x))$ for $i\le l-1$, then $lc(\bar{r}_i)=cont(r_i(x))$.

For the second part, according to the assumption, $\gcd(lc(r_i),cont(r_{i+1}(x)))=cont(r_i(x))$, then $cont(r_i(x))|cont(r_{i+1}(x))$. Also $\bar{r}_i(x)=\psi r_i(x)+\phi\bar{r}_{i+1}(x)x^{\delta_i}$, so $cont(r_i(x))|\bar{r}_i(x)$. Due to $lc(\bar{r}_i)=cont(r_i(x))$, we know that $lc(\bar{r}_i)=cont(r_i(x))|\bar{r}_i(x)$, for $0\le i\le l$.

Lemma 5. Let $r_1(x),\cdots,r_l(x)$ be the polynomial remainder sequence obtained in improved resultant algorithm. Then $\gcd(lc(r_i),cont(r_{i+1}(x)))=cont(r_i(x))$.
Proof. We prove this lemma by induction on i, $i=1,\dots,l-1$.

First, suppose that $\boldsymbol{H}$ is the Hermite Normal Form of the ideal lattice $\mathcal{L}$ generated by $g(x)\in\mathbb{Z}[x]/\langle f(x)\rangle$, and $r_i(x)$ belongs to $\mathcal{L}$. Denote $\frac{lc(r_i)}{lc\left(H_{n-n_i}\right)}$ as γ_i and $d_i=n-n_i$, here $\boldsymbol{H}_i(x)$ is the corresponding polynomial of the i-th row, then $r_i(x)=\gamma_i\boldsymbol{H}_{d_i}(x)+\sum_{j=i+1}^{l}A_{i,j}(x)\,\boldsymbol{H}_{d_j}(x)$, where $\deg(A_{i,j})<n_i-n_j$.

From Lemma 4, $lc\left(\boldsymbol{H}_{d_j}\right)|\boldsymbol{H}_{d_j}(x)$, for $i \le j \le l$. So $lc\left(\boldsymbol{H}_{d_i}\right)|cont\left(r_i(x)\right)$. Because $r_i(x) = t_i(x)g(x)modf(x)$ belongs to $\mathcal{L}$ and $t_i(x)$ is primitive, then $\gcd(\gamma_i, A_{i,i+1}(x), \dots, A_{i,l}(x)) = 1$, thus there exists some $i < k \le l$, $cont\left(\frac{r_i(x)}{lc\left(H_{d_i}\right)}\right) = \gcd(\gamma_i, \frac{lc\left(H_{d_k}\right)}{lc\left(H_{d_i}\right)})$, which means that $cont\left(r_i(x)\right) = \gcd(lc(r_i), lc(H_{d_k}))$. So every content of $r_i(x)$ must be a factor of $\boldsymbol{H}_n$. Specially, we have $cont\left(r_{l-1}(x)\right) = lc(\bar{r}_{l-1})$, which shows that the result holds for $i = l - 1$.

Now assume that for $i \ge k$, we have $\gcd(lc(r_i), cont\left(r_{i+1}(x)\right) = cont(r_i(x))$. Then from Lemma 3, we have $lc(\bar{r}_i) = cont\left(r_i(x)\right)$.

Next we consider $k - 1$, from the Algorithm 2, $\gcd(lc(r_{k-1}), lc(\bar{r}_k)) = lc(\bar{r}_{k-1})$. Then because $lc(\bar{r}_i) = cont\left(r_i(x)\right)$ for $i \ge k$, $\gcd(lc(r_{k-1}), cont\left(r_k(x)\right)) = lc(\bar{r}_{k-1})$. So we need to show $lc(\bar{r}_{k-1}) = cont\left(r_{k-1}(x)\right)$.

First, $cont\left(r_{k-1}(x)\right)|lc(r_{k-1})$ and according to the Lemma 3, $cont\left(r_{k-1}(x)\right)|cont(\bar{r}_k(x))$, so $cont\left(r_{k-1}(x)\right)|\gcd(lc(r_{k-1}), cont\left(r_k(x)\right)) = lc(\bar{r}_{k-1})$. We suppose $lc(\bar{r}_{k-1}) = a \cdot cont(r_{k-1}(x))$ for a prime a. According to Lemma 4, $lc(\bar{r}_i) = cont\left(r_i(x)\right) = cont\left(\bar{r}_i(x)\right)$ for $i \ge k$, so we have $cont\left(r_{k-1}(x)\right)|cont(\bar{r}_{k-1}(x))$. Also the step diminishes the leading coefficient and $lc(\bar{r}_{k-1})|lc(r_{k-1})$, then $cont(\bar{r}_{k-1}(x)) \le cont(r_{k-1}(x))$. So $cont\left(r_{k-1}(x)\right) = cont(\bar{r}_{k-1}(x))$.

Consider the$k$-th equation in improved subresultant algorithm,

$$\alpha_k r_{k-2}(x) = q_k(x) r_{k-1}(x) + \beta_k r_k(x)$$

here $\alpha_k = lc(r_{k-1})^{\delta_{k-2}+1}$. Because $a \cdot cont\left(r_{k-1}(x)\right)|lc(r_{k-1})$ and $a \cdot cont\left(r_{k-1}(x)\right)|cont\left(r_k(x)\right)$, we know that $a \cdot \left(cont\left(r_{k-1}(x)\right)\right)^{\delta_{k-2}+1}|\alpha_k$ and $a \cdot cont\left(r_{k-1}(x)\right)|r_k(x)$, which means, if we divide the equation above by $\mu = a \cdot \left(cont\left(r_{k-1}(x)\right)\right)^{\delta_{k-2}+1} cont(r_{k-2}(x))$, then $\frac{\alpha_k r_{k-2}(x)}{\mu}$and $\frac{\beta_k r_k(x)}{\mu}$ both belong to $\mathbb{Z}[x]$, while $\frac{q_k(x) r_{k-1}(x)}{\mu}$ doesn't. This is a contradiction. So the proof is completed.

Using the results above, we realize that $lc(t_i)$ is related to τ_i which is the unknown. We tried some equations and found the following equation,

$$\gcd\left(lc(t_i), lc(\bar{r}_{i-1})\right) = gcd\left(\frac{lc(r_{i-1})}{lc(\bar{r}_{i-1})}, lc(\bar{r}_{i-1})\right)$$

for $i = 0,1,2,\cdots,l$.Also in our experiments, the conjecture hold with extremely high probability.

4. A New Algorithm for Computation of Polynomial Greatest Common

In this section, we give a probabilistic subresultant algorithm by applying the results in the last section. We need to emphasis that the algorithm is not deterministic yet. The detail of the algorithm is present as following.

Algorithm 3 Probabilistic Subresultant Algorithm

Input: two polynomials $f(x)$, $g(x) \in \mathbb{Z}[x]$ with degree n and m respectively and $f(x)$ is monic and irreducible

Output: Probabilistic subresultant PRS, $r_0(x), r_1(x), \cdots$

1.[Initialize] $l \leftarrow h \leftarrow 1, u_1(x) \leftarrow f(x), u_2(x) \leftarrow g(x), i \leftarrow 1$

2.Compute $lc(u_2)^{\delta+1}u_1(x) - q(x)u_2(x) = r(x)$, here $deg(r) < deg(u_2)$, $\delta = deg(u_1) - deg(u_2)$

3.$u(x) \leftarrow u_1(x)$, $u_1(x) \leftarrow u_2(x)$, $u_2(x) \leftarrow r(x)$

4.When $deg(u_2) \neq 0$,

4.1 $l \leftarrow lc(u_2)$, $h \leftarrow l^{\delta}h^{1-\delta}$

4.2 $\tau \leftarrow \gcd(h, cont(u_2(x)))$, $\tau' \leftarrow \gcd(lc(u)/cont(u(x)), cont(u(x)))$

4.3 $\tau \leftarrow /\tau'$, $r_i(x) \leftarrow r(x)/(lh^{\delta}\tau)$

4.4 $\delta = deg(u_1) - deg(u_2)$

4.5 Compute $lc(u_2)^{\delta+1}u_1(x) - q(x)u_2(x) = r(x)$, $deg(r) < deg(u_2)$, $\delta = deg(u_1) - deg(u_2)$

4.6 $u(x) \leftarrow u_1(x)$, $u_1(x) \leftarrow u_2(x)$, $u_2(x) \leftarrow r(x)/(lh^{\delta})$

4.7 $i \leftarrow i + 1$

5.[Return] $r_0(x), r_1(x), \cdots$

For the often-used polynomials in ideal lattice-based cryptography $x^n + 1$ and $x^n - x - 1$, here n is a power of 2, we give the experiment results. For each polynomial, we sample 10000 examples randomly with coefficients in the range [-20,20] and the correctness is present below.

Polynomial	$x^n + 1$	$x^n - x - 1$
Correctness	97.88%	99.73%

5. CONCLUSIONS

In this paper, we give some results about the contents and small factors of remainders during Euclidean algorithm of polynomials. By applying these results, we proposed a probabilistic subresultant which can output correct remainders with high probability.

Due to the case of failure, the next research will be focus on the exact expression of each τ_i and relation between $lc(t_i)$ and $\mathrm{cont}(r_j(x))$ to obtain a determinisitic improved subresultant algorithm.

REFERENCES

[1] G.E. Collins. Subresultants and reduced polynomial remainder sequences. J. ACM, 14(1): 128--142 (1967)

[2] D.E. Knuth. The art of computer programming. Seminumerical Algorithms. vol. 2, 3rd Edition, 1998 (1st Edition, 1969)

[3] W.S. Brown. The subresultant PRS algorithm. ACM Trans. Math. Software, 4(3):237--249 (1978)

[4] Y. Zhang, R.Z. Liu, D.D. Lin. Fast Triangularization of Ideal Latttice Basis. Journal of Electronics and Information Technology, 42(1): 98-104 (2020)

[5] Y. Zhang, R.Z. Liu, D.D. Lin. Improved Key Generation Algorithm for Gentry's Fully Homomorphic Encryption Scheme. ICISC: 97-111 (2018)

[6] J Gathen, T Lücking. Subresultants Revisited. Latin American Symposium on Theoretical Informatics, LNCS 1776: 318–342