ISSN: 2424-8460(O)
2251-2608(P)

Original Research Article

A more efficient algorithm for Chinese remainder theorem of sparse polynomials

Mingjun Ouyang
Keystone Academy, Beijing, 101300, China

Abstract: The Chinese remainder theorem (CRT) for polynomials is an extension of the well-known theorem in number theory, which deals with the existence and uniqueness of solutions to a system of congruences. One formulation of CRT involves recovering an unknown polynomial f given its reductions modulo a se- quence of polynomials g_i , i = 1, 2, ...,s. In the classical setting, the polynomials g_i are required to be coprime, and the sum of their degrees $\sum_{i=1}^{s}$ deg g_i must exceed degf. In this paper, we present a sparse algorithm tailored for the case where the number of terms in f is no more than T, and the polynomials g_i have the form $g_i = x^{pi}-1$. Our algorithm demonstrates that under these conditions, the sum of the degrees of g_i needs only satisfy $\sum$deg $g_i \geq \tilde{O}(T^2 \log^2 D)$ to uniquely determine f, where degf is at most D. This smaller degree sum con- dition provides an advantage over traditional Polynomial Chinese Remainder Theorem approaches when $T^2 \log^2 D < D$, particularly when f is sparse. Our sparse algorithm thus offers an efficient way to recover sparse polynomials.
Keywords: Chinese remainder theorem; sparse polynomial; interpolation

1. Introduction

Chinese Remainder Theorem (CRT) is a fundamental result in number theory and can be used in many areas with many extensions. Especially, the problem of interpolating sparse polynomials using CRT has a high importance because of its generality. Sparse polynomials, characterized by having a relatively small number of nonzero terms compared to their degree, algorithms related to them have significant challenges in terms of efficient interpolation and reconstruction.There are a lot of pioneers who investigated the algorithm before. For example, Giesbrecht and Roche[4] in 2011 introduced an algorithm that is random for interpolating multivariate polynomials in finite fields with fewer evaluations, and provided the first absolute (not probable) method for sparse polynomials with complex coefficients without relying on root finding. Moreover, Arnold et al. [1] in 2014 proposed a Monte Carlo algorithm that improves interpolation of sparse polynomials in finite fields by leveraging coefficient information, enhanc- ing efficiency over previous methods. Also, Arnold et al. [2] in 2015 presented an algorithm for efficiently representing polynomials from straightline programs, improving the algorithm in cases with many variables and big degrees.

The method proposed by Garg and Schost[3] is the most significant pioneer in this algorithm that relates to our new algorithm. They developed a deterministic algorithm that extends the CRT to interpolate sparse polynomi- als, achieving a complexity of $O(T^2 \log D)$ for the number of reductions required and $\tilde{O}(T^2 \log D)$ for the degree needed for reductions, where T is the number of nonzero terms and D is the degree of the polynomial. While this approach represented a significant advancement, it leaves room for further optimization in terms of computational efficiency.

This paper presents an improved method that reduces both the number of reductions needed and the degree of reductions to $O(T \log D)$ and $\tilde{O}(T \log D)$ respectively. By changing the selection process for reductions and optimizing the reduction steps, our method offers a more efficient solution to solve the original polynomial.

ISSN: 2424-8460(O)
2251-2608(P)

2. Previous work

2.1. Classical algorithm for CRT of polynomials

The classical algorithm for CRT of Polynomials is exactly the same as the one of integers. The polynomial f(x) can be reduced using the formula:

$$f(x) = \sum_{i=1}^{s} a_i(x) \times G_i(x) \times y_i(x) \bmod G(x)$$

where:

a_i (x) is the residue off(x) modulo g_i (x),

$G_i(x) = \frac{G(x)}{g_i(x)}$ is the partial product of the moduli,

$y_i(x)$ is the modular inverse of $G_i(x)$ modulo $g_i(x)$,

$G(x) = g_1(x) \times g_2(x) \times \cdots \times g_s(x)$ is the product of all moduli.

The total time complexity of the CRT algorithm for polynomials, considering both the number of moduli and the degrees of those moduli, is: O(Dlog D)

where $D = \sum_{i=1}^{s} degg_i$ is the sum of the degrees of the individual moduli.From this, we can see that the complexity of CRT algorithm is closely related to $\sum_{i=1}^{s} degg_i$.

2.2. Gargschost algorithm

The method proposed by Garg and Schost [3] employs the Chinese Remainder Theorem in conjunction with straightline programs to interpolate sparse polynomials. Their deterministic algorithm is characterized by its use of a relatively large number of prime numbers and a large degree of reductions to ensure the accurate reconstruction of the polynomial.

Their original method sets bounds on the number of terms (T), the degree of the polynomial (D), and the size of the straightline program (L). The complexity of their algorithm is $O\tilde{}(LT^4 \log^{2(D)})$ for univariate polynomials. They use evaluations at roots of unity and symmetric functions to determine the structure of the polynomial. This approach enables the recovery of the polynomial's terms even when they are provided in a complex, compact form via the straight-line program. After evaluating the polynomial modulo several $(x^{pi} - 1)$, the method reconstructs the exponents from the reductions. This helps in deterministically finding the exponents. However, the drawback of this algorithm is that it is too large in the number of reductions and in the degree of reductions needed. Specifically, their approach in Chinese Remainder Theorem requires $s=O(T^2 \log D)$ and $degg_i=O\tilde{}(T^2 \log D)$, where T denotes the number of nonzero terms in the polynomial and D denotes its degree. This method effectively addresses the interpolation problem but does so at the cost of increased computational resources if T increases.

Preliminary Result-Previous Chinese Remainder Theorem Given a polyno- mial $f \in Z[x]$ and a list of polynomials $g_1, g_2, ..., g_s \in Z[x]$. Assume g_i's are pairwise coprime.

Then we have the Chinese Remainder Theorem.

Theorem 1. [5] f can be reconstructed from the reductions f mod g_i, i=1, ...,s if $\sum_{i=1}^{s} degg_i > degf$.

In the above theorem, the sum of the degrees of g_i needs to be greater than

the degree off in order to recover f. That is, $\sum_{i=1}^{s} =1\ degg_i > degf$.

Problem .1. What we want to explore is whether it is possible to select special

gi 's so that $\sum_{i=1}^{s} degg_i < degf$ can still recover f.

This article tells us that in special situation, it is possible. The special meaning here is that f is sparse, and then gi is selected as the form $g_i = x^{pi} - 1$, where p_i is a prime.

In order to achieve the above objectives, the direction of improvement is:

Reduce the number s, which is the number of g_i needed.

Reduce the degree $degg_i$, which is the degree of g_i needed.

ISSN: 2424-8460(O)
2251-2608(P)

2.3. Theoretical comparison

We list the comparison between our algorithm and previous algorithms in the table below.

Table 1. Comparison of the efficiency of previous methods and clas- sical algorithm with ours.

	Number of Reductions (s)	Degree of Reductions ($degg_i$)	$\Sigma_{i=1}^{s} degg_i$
Classical Algo- rithm	*	*	O(D)
Garg-Schost's Work	$O(T^2 \log D)$	$O^{\sim}(T^2 \log D)$	$O^{\sim}(T^4 \log^2 D)$
Our Work	O(Tlog D)	$O^{\sim}$ (Tlog D)	$O^{\sim}(T^2 \log^2 D)$

Here* in classical algorithm indicates that the degrees of gi can be dis- tributed unevenly, as long as $\Sigma_{i=1}^{s} degg_i > D$ is satisfied. In both the GargSchost 's algorithm and our algorithm, $\Sigma_{i=1}^{s} degg_i$ is in $O(s \times \max\{degg_i, i = 1,..., s\})$, which is the number of reductions multiplied by the degree of reduc- tions.

Compared to Garg-Schost's work, the theoretical improvements can be sum- marized as follows:

Number of reductions: reduce s from $O(T^2 \log D)$ to O(Tlog D).

Degree of reductions: reduce deggi from $O^{\sim}(T^2 \log D)$ to $O^{\sim}$ (Tlog D).

The total $\Sigma_{i=1}^{s} degg_i$ inGarg-Schost's work is T^2 times bigger than ours.

Compared to classical algorithm, because we are finding the algorithm for spares polynomial, T will be smaller than D. Moreover, logarithm itself has a relatively lower magnitude than linear. Therefore, as long as $T^2 \log 2\ D < D$, our method is more efficient, and it is more efficient in a lot of cases because of the reasons above(Smaller T than D due to spares polynomial and small magnitude of log D). Thus, our algorithm in total degree translate to significant computational efficiency, particularly for polynomials with a large degree D and a moderate number of non-zero terms T.

2.4. Practical implications

In practical terms, the lower total degree $\Sigma_{i=1}^{s} degg_i$ means that fewer computational resources are required for polynomial interpolation. This makes our approach more suitable for applications involving large-scale symbolic computations, where efficiency is critical. Additionally, the deterministic nature of our algorithm ensures consistent performance and reliability, which is crucial for applications in cryptography and coding theory.

3. Preliminaries

For a polynomial $f(x) = c_1 x^{e1} + ... + c_t x^{et} \in Z[x]$, assume $D \geq degf$ and denote #f as the number of terms in f. Denote $f_p := f \bmod (x^p - 1)$ as the remainder of f divided by $x^p - 1$, therefore we have $f_p = c_1 x^{e1 \bmod p} + ... + c_t x^{et \bmod p} \in Z[x]$.

Definition .1. For a term $c_i x^{ei}$ in f, we say it collides in f_p if there exists another term $c_j x^{ej}$ in f such that $x^{ei} \bmod (x^p - 1) = xej \bmod (x^p - 1)$, or in other words, $p|(e_i - e_j)$.

If $c_i x^{ei}$ does not collide in f_p, then we say p is an OK prime for $c_i x^{ei}$, otherwise p is a bad prime for $c_i x^{ei}$.

For a polynomial $f(x) = c_1 x^{e1} + ...+ c_t x^{et}$ with ci≠0. Denote $Terms(f) = \{c_1 x^{e1}, ..., c_t x^{et}\}$.

4. A new algorithm–sparse Chinese remainder theorem

In order to find a more efficient algorithm for sparse Chinese Remainder Theorem, the Garg-Schost's algorithm needs to be introduced [3]. In their algorithm, the number of reductions needed is $O(T^2 \log D)$ and the degree of the reductions are also $O^{\sim}(T^2 \log D)$.

In a probability algorithm [4], the number of reductions is reduced to O(Tlog D), but the degree of reductions are still $O(T^2 \log D)$.

In our new deterministic algorithm, the purpose is to decrease the number of reductions needed from $O(T^2 \log D)$ to O(Tlog D) and the degree of reductions also from $O^{\sim}(T^2 \log D)$ to $O^{\sim}$ (Tlog D).

ISSN: 2424-8460(O)
2251-2608(P)

4.1. Main idea

Assume that $f(x) = c_1x^{e1} + ... + c_tx^{et}$ and $p_1,... , p_N$ are different primes, and f_{pi} is the reductions of f divided by $x^{pi} - 1$.

Suppose $f_{p1},..., f_{pN}$ are given and we try to find the exact form of f from those reductions.

Without losing of generality, we recover the term c_1x^{e1} as an example. In order to find it, we need to make sure it does not collide in most of fpi 's.

We calculate how many reductions fpi is needed. For a list of different primes $q_1,...,q_K$, if c^1x^{e1} and c^2x^{e2} can be combined together after mod $(x^{qi} - 1)$, then it means qi $|(e_1 - e_2)$, i = 1,..., K As qi are distinct primes and $q_i \geq 2$, $2^K \leq q_1 \cdot q_2 ... q_K \leq |e_1 - e_2| \leq D$. Therefore, there are at most $K \leq \log_2 D$ bad primes that will make c_1x^{e1} and c_2x^{e2} collide.

Then consider the first term and the third term, c_1x^{e1} and c_3x^{e3} , there are also $K \leq \log_2 D$ bad primes that will make them collide. This pattern continues to the last one, c_1x^{e1} and c_tx^{et}. Therefore, for c_1x^{e1}, the number of bad primes that make it collide is

$$B = \lceil (t - 1)\log_2 D \rceil$$

Thus, in $f_{p1},..., f_{pN}$, there are at most B reductions where c_1x^{e1} collided. So the number of reductions that c_1x^{e1} does not collide is N − B.

Add B by 1 and let

$$N = B + 1 = \lceil (t - 1)\log_2 D \rceil + 1,$$

it will at least give us one OK prime for c_1x^{e1} in $p_1,... ,p_N$.

Given

$$f \mod (x^{p_i} - 1), i = 1, . . . ,N,$$

where $p_1,..., p_N$ are distinct prime . Because of the above analysis, c_1x^{e1} does not collide in at least one off_{pi}. Similarly, c_2x^{e2} off, and so are all the rest of the terms in f.

However, even if we know c_1x^{e1} does not collide in fp and $c_1x^{e1 \bmod p}$ is the corresponding term in f_p , we still cannot recover c_1x^{e1} from $c_1x^{e1 \bmod p}$. There- fore, in order to find c_1x^{e1} in f that does not collide, we can start to add more primes in. As above analysis, in $f_{p1},...,f_{pB+1}$, there is 1 reductions where c_1x^{e1} does not collide, then if a f_{pB+2} is added there will be 2 reductions where c1 xe1 does not collide. When keeping on adding the reductions to $f_{p1},..., f_{p2B+1}$, there will be B + 1 reductions where c_1x^{e1} does not collide, which exceeds half. If we can identify which terms are collided or which are not, we can recover the original polynomial f. Take 2B + 1 primes, then the primes are $p_1,... , p_{2B+1}$ and the reductions after modulo is $f_{p1},..., f_{p2B+1}$.

Take any term cx^d , we need to determine whether it is a term in f. We have the following theorem.

Theorem 2. Suppose cx^d is a term and $f_{p1}, f_{p2},..., f_{p\,(2E+1)}$ are 2E + 1 re- ductions of f with form f_{pi}= f mod $(x^{pi} - 1)$. Here E = ⌈ T $\log_2$ D ⌉ and T ≥ #f, D ≥ degf and pi are different primes. Let S = $\{p_i \,|cx^{d \bmod pi} \in Terms(f_{pi})$ and i = 1, ..., 2E + 1$\}$. Then if $cx^d \in$ Terms(f), #S ≥ E + 1; if cxd ∉ Terms(f), #S ≤ E . Proof. We consider three cases.

• If $cx^d \in$ Terms(f), then cx^d is a term of f. If cx^d does not collide in f_{pi}, then $cx^{d \bmod pi} \in Terms(f_{pi})$. As stated above, there are at most B (see Eq. (1)) reductions in $f_{p1},..., f_{p\,(2E+1)}$ where cx^d collides, therefore #S ≥ 2E + 1 − B ≥ E + 1.

• If $cx^d \notin$ Terms(f) and d ∉ $\{e_1,..., e_t\}$, let $\hat{\mathbf{f}} = f + cx^d$. Then # = #f +1. If cxd does not collide in pi , then $cx^{d \bmod pi} \in Terms(\hat{\mathbf{f}}_{pi})$. Thus $cx^{d \bmod pi} \notin Terms(f_{pi})$, that is $\hat{\mathbf{f}}_{pi} \notin$ S. As the number of this kind of p_i is at least E + 1. Thus in this case, #S ≤ 2E + 1 − (E + 1) = E.

• If $cx^d \notin$ Terms(f) and d ∈ $\{e_1,..., e_t\}$, let $\hat{\mathbf{f}} = f - cx^d$. Then #$\hat{\mathbf{f}}$ = #f. Without loss of generality, assume e_1 = d. Then $(c_1 - c)x^d$ is a term in . If $(c_1 - c)x^d$ does not collide in pi , then $(c_1 - c)x^{d \bmod pi} \in Terms(\hat{\mathbf{f}}_{pi})$, and then $c_1x^{d \bmod pi} \in Terms(f_{pi})$. Therefore, $cx^{d \bmod pi} \notin Terms(_{fpi})$.

ISSN: 2424-8460(O)
2251-2608(P)

As the number of this kind of p_i is at least B + 1, that is $p_i \notin S$. Thus in this case, $\#S \leq 2E + 1 - (B + 1) = E$.

Once a term cx^d is found, update f:= f − cx^d and the total number of terms will reduce by one. Keep doing this till to f = 0, the function f can be recovered. As E = ⌈ T $\log_2$ D⌉, B =⌈ (t-1)$\log_2$D⌉, the magnitude of E and B are still O(Tlog D).

Since we only increase B to 2E+1, which have no effect of the overall magnitude. Comparing to Algorithm in Garg and Schost's algorithm [3], whose magni- tude is $O(T^2 \log D)$, if f has more than two terms, the new calculation has a bigger advantage on the efficiency. As the magnitude of i-th prime is O˜ (i), OK prime has magnitude of O~ (Tlog D). Comparing with the good prime in Garg and Schost's paper [3] whose magnitude is $O(T^2 \log D)$, the magnitude of the OK prime numbers are smaller.

We summarize the above analysis into Algorithm 1.

Algorithm 1 More Efficient Chinese Remainder Theorem for Sparse Polyno- mial

Require:
1. Different primes p_i, i = 1, ... , 2E + 1 with E = ⌈ T $\log_2$ D⌉, where T ≥ #f, D ≥ degf.
2. Reductions f_{pi} = f mod (x^{pi} − 1), i = 1, ... , 2E + 1.

Ensure: The exact form of f.
1. Let g := 0;
2. Collect all coefficients of f_{pi}, i = 1, ..., 2E + 1 and put them into a set C.
3. repeat
4. for d = 0,...,D and for each c ∈ C
5. if cx^d satisfies the condition of Theorem 2 that #S ≥ E + 1; then g := g + cx^d and update fpi := f_{pi} − cx^d modp, i = 1, . . . , 2E + 1.
end if
6. until $\#f_{pi} = 0$, i = 1, ..., 2E + 1.
7. return g.

Theorem 3. The number of reductions in Algorithm 1 is O(Tlog D) and the degree of reductions is O˜ (Tlog D) .

Proof. Firstly, the number of reductions in Algorithm 1 is obvious. We choose a total of 2E +1 prime numbers, where E = ⌈ T $\log_2$ D⌉. Therefore, the number of reductions is O(Tlog D).

The degree of reduction is also true because f_{pi}= f mod (x^{pi}−1), which means that degree of f_{pi} will not exceed p_i. According to Modern Computer Algebra [5], the i-th prime will have magnitude O˜ (i). Therefore the degree of reduction is O˜ (Tlog D) because we have 2E + 1 primes and E = ⌈ T $\log_2$ D⌉.

Theorem 4. Algorithm 1 works correctly as specified.

Proof. We need to make sure the output g is equal to f. To prove it, we need to prove that all cx^d in Step 5 that satisfy the condition of Theorem 2 that #S ≥ E + 1 are all the terms in f.

Firstly, all coefficients in f appear in set C because we use 2E +1 reductions to find the polynomial. For every term cx^d of f, the coefficient c will appear in the coefficients of reductions f_{p1},..., f_{p2E+1} , as B < 2E + 1. Therefore, all coefficients of f will enter in C in Step 2.

Secondly, all terms in f will appear in step 5 because all terms in f are constructed from a coefficient and a degree ≤ D. In step 2, all coefficients of f are in set C; all degrees are ≤ D. Therefore, combining values in C and d will give all terms in f, but with extra terms that are not in f.

Thirdly, because there are extra terms that are not in f appearing when combining values in C and d to form terms, step 5 uses Theorem 2 to take out all terms that are obeying the first condition. Therefore, what is left are only and all terms in f.

Since g is the addition of all cx^d, and cx^d s are all and only terms in f , meaning g 三 f. We proved it.

5. Conclusion

In conclusion, this paper focuses on finding a more efficient Chinese Remainder Theorem algorithm for recovering a sparse polynomial. Using Garg-Schost's work as a base and guide, we are trying to reduce a polynomial f that is no more than T terms from its reductions modulo a sequence of polynomials g_i= x_{pi}−1. We increase the efficiency by decreasing the number of reductions from O(T2 log D) to O(Tlog D) and decreasing the degrees of reductions from O˜ (T_2 log D) to O˜ (Tlog D). Thus, our algorithm contributed to the improvement

ISSN: 2424-8460(O)
2251-2608(P)

of a more efficient algorithm for recovering a sparse polynomial.

Acknowledgements

Because this thesis was created entirely by me alone. I didn't get any more help or guidance from anywhere other than my tutor and the literature. I chose my topic solely out of my own research and interest. I learned about the basic Chinese remainder theorem from my study of the competition, and thus devel- oped an interest in researching topics in the related direction. In determining my topic, I initially explored more aspects of the Chinese Remainder Theorem, including the way it is used in recovering polynomials. I then conducted more relevant searches and found the study of Garg-Schost. They specialize in deter- ministic algorithms that are more efficient when recovering sparse polynomials. And, I read another study that explored probabilistic algorithms for the same problem. I found that the probabilistic algorithm was more efficient, but it was not 100% successful, whereas the deterministic algorithm was more com- putationally complex, but 100% successful. Therefore, I then wanted to go and find a more efficient deterministic algorithm. During my derivation, my mentor gave me some help and guided me further at many points, such as deciding the number of primes and how to determine whether each term collides after modulo operation. The difficulties I encountered during the derivation were relatively few and generally worth the amount of time I spent thinking about them. While writing the paper, I was sometimes unfamiliar with the use of LATEX, for example, I did not know how to type symbols like ~ and symbols for modulo operations. Therefore, my tutor helped me and taught me how to use LATEX more skillfully in these areas, and in the end I was able to present them accurately.

References

[1] A. Arnold, M. Giesbrecht, and D. S. Roche. Sparse interpolation over finite fields via low-order roots of unity. In Proceedings of the 39th International Sym- posium on Symbolic and Algebraic Computation, ISSAC '14, page 27–34, New York, NY, USA, 2014. Association for Computing Machinery.

[2] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse multivariate polyno- mial interpolation of straight-line programs. Journal of Symbolic Computa- tion, 75:4–24, 2016. Special issue on the conference ISSAC 2014: Symbolic computation and computer algebra.

[3] S. Garg and . Schost. Interpolation of polynomials given by straight-line programs. Theoretical Computer Science, 410(27-29):2659–2662, 2009.

[4] M. Giesbrecht and D. S. Roche. Diversification improves interpolation. In Pro- ceedings of the 36th International Symposium on Symbolic and Algebraic Computation, ISSAC '11, page 123–130, New York, NY, USA, 2011. Asso- ciation for Computing Machinery.

[5] J. von zur Gathen and J. Gerhard. Modern computer algebra. Cambridge University Press, USA, 1999.