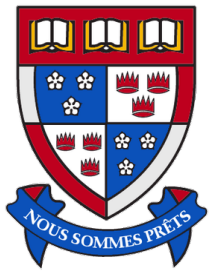
Optimizing the Fast Euclidean Algorithm in $\mathbb{Z}_p[x]$

Michael Monagan Garrett Paluck

Department of Mathematics, Simon Fraser University, British Columbia

April 17, 2024



Optimizing the Fast Euclidean Algorithm in $\mathbb{Z}_p[x]$

We have implemented an optimized version of the Fast Euclidean Algorithm in C using the Half Gcd algorithm from [GJ13] to calculate the gcd of two polynomials over $\mathbb{Z}_p[x]$ for some prime p.

The work is motivated by Huang in [HM24] who reduces the calculation of the sparse gcd of two polynomials in $\mathbb{Z}[x_1, ..., x_n]$ to many gcd calculations in $\mathbb{Z}_p[x_1]$.

Some of the optimizations we have implemented are:

- using FFT multiplication for high degree polynomials.
- minimizing the total number of uses of the FFT algorithm.
- Using the extended Euclidean algorithm for low degree recursive calls to the Half Gcd algorithm.

Minimizing the total number of the FFT calls

Let $f \in F[x]$ where $\deg(f) < n = 2^k$ and let $\omega \in F$ be a primitive nth root of unity. We define the Discrete Fourier Transform (DFT) as:

$$\mathrm{DFT}_\omega : f \mapsto \left(f(1), f(\omega), f(\omega^2), ..., f(\omega^{n-1})\right)$$

Let $A, B, C \in F[x]^{2\times 2}$ s.t.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \text{ and } C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

We want to optimally calculate $C = A \times B$ using FFT multiplication. For simplicity, let's assume the degree of every polynomial in A, B and C is less than $n = 2^k$.

Minimizing the total number of the FFT calls

We use the calculation of polynomial c_{11} to illustrate the optimization

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Inefficient Method:

$$c_{11} = \tfrac{1}{n}\mathrm{DFT}_\omega^{-1}(\mathrm{DFT}_\omega(a_{11}) \cdot \mathrm{DFT}_\omega(b_{11})) + \tfrac{1}{n}\mathrm{DFT}_\omega^{-1}(\mathrm{DFT}_\omega(a_{12}) \cdot \mathrm{DFT}_\omega(b_{21}))$$

Each polynomial in C requires 6 FFT calls (24 total)

Efficient Method:

Precompute DFT_ω of each polynomial in A and B once (8 total)

The DFT_ω is a linear transformation. That is

$$\mathrm{DFT}_\omega(f + g) = \mathrm{DFT}_\omega(f) + \mathrm{DFT}_\omega(g)$$

$$c_{11} = \tfrac{1}{n}\mathrm{DFT}_\omega^{-1}(\mathrm{DFT}_\omega(a_{11}) \cdot \mathrm{DFT}_\omega(b_{11}) + \mathrm{DFT}_\omega(a_{12}) \cdot \mathrm{DFT}_\omega(b_{21}))$$

This method allows us to compute C using 12 total FFT calls.

Gcd Timings

We calculated $\gcd(a, b)$ for two randomly generated polynomials $a, b \in \mathbb{Z}_p[x]$ for $p = 2^{28} + 2097153$.

deg a/deg b	Our algorithm	Maple Gcdex	Maple GCD	Euc. Alg. in C	Magma
$2^6 - 1/2^6 - 2$	0.0002	0.0001	0.0001	0.0002	0.0002
$2^7 - 1/2^7 - 2$	0.0004	0.0003	0.0001	0.0004	0.0003
$2^8 - 1/2^8 - 2$	0.0009	0.0010	0.0004	0.0009	0.0003
$2^9 - 1/2^9 - 2$	0.0022	0.0038	0.0016	0.0024	0.0012
$2^{10} - 1/2^{10} - 2$	0.0051	0.0140	0.0061	0.0071	0.0032
$2^{11} - 1/2^{11} - 2$	0.0122	0.0544	0.0240	0.0243	0.0079
$2^{12} - 1/2^{12} - 2$	0.029	0.245	0.092	0.081	0.0195
$2^{13} - 1/2^{13} - 2$	0.068	0.965	0.365	0.326	0.049
$2^{14} - 1/2^{14} - 2$	0.156	3.834	1.454	1.365	0.120
$2^{15} - 1/2^{15} - 2$	0.349	15.176	5.823	5.237	0.320
$2^{16} - 1/2^{16} - 1$	0.787	60.165	23.289	21.149	0.800
$2^{17} - 1/2^{17} - 1$	1.764	239.876	93.172	82.512	1.960
$2^{18} - 1/2^{18} - 1$	3.967	956.839	373.092	329.318	4.700
$2^{19} - 1/2^{19} - 1$	8.838	-	-	-	11.100
$2^{20} - 1/2^{20} - 1$	19.403	-	-	-	26.100

References

Joachim von zur Gathen and Gerhard Jürgen, *Modern computer algebra*, 3 ed., Cambridge University Press, 2013.

Qiao-Long Huang and Michael Monagan, *A new sparse polynomial gcd algorithm by separating terms*, Submitted to ISSAC 2024 (2024).