Sparse polynomial arithmetic

Bruno Grenet[1]

LIRMM – Université de Montpellier

LJK, Grenoble – February 10., 2022

[1]Based on joint works with P. Giorgi, A. Perret du Cray and D. Roche

Dense and sparse polynomials

$$f = \sum_{i=0}^{D} c_i x^i \qquad c_i \in R$$

Dense representation

- List of all the c_i's
- Arithmetic size: $O(D)$
- Bit size: $O(D \log H)$

Basic operations

- Addition: $O(D)$ (trivial)
- Multiplication : $\tilde{O}(D)$ *via* FFT
- Euclidean division, GCD, multipoint evaluation, interpolation, … $\rightarrow \tilde{O}(D)$ *via* reduction to multiplication

Sparse representation

- List of the pairs (i, c_i) for **nonzero** c_i's
- Arithmetic size: $O(T)$ (*sparsity of* f)
- Bit size: $O(T(\log D + \log H))$

Basic operations

- Addition: $O(T)$ (list merge)
- Multiplication, division, GCD, …: ?

Multivariate polynomials

$$f = \sum_{i=0}^{T} c_i x_1^{e_{1i}} x_2^{e_{2i}} \cdots x_n^{e_{ni}}$$

Sparse representation

- List of the tuples $(e_{1i}, \ldots, e_{ni}; c_i)$
- Bit size: $O(T(n \log D + \log H))$ where $D = 1 + \max_{i,j} e_{ji}$

Kronecker substitution

- $f \mapsto f_u(x) = f(x, x^D, x^{D^2}, \ldots, x^{D^{n-1}}) = \sum_i c_i x^{e_{1i} + e_{2i} D + \cdots + e_{ni} D^{n-1}}$
- f_u has degree D^n $\rightarrow$ same bit size as f
- Invertible transformation, compatible with polynomial operations

Work with univariate polynomials – same results for multivariate polynomials

1. Known algorithms and challenges

2. The main tool: sparse interpolation

3. Fast multiplication and exact division algorithms

1. Known algorithms and challenges

2. The main tool: sparse interpolation

3. Fast multiplication and exact division algorithms

Multiplication of sparse polynomials

$$\left(\sum_{i=0}^{T-1} c_i x^{e_i}\right) \times \left(\sum_{j=0}^{T-1} d_j x^{f_j}\right) = \sum_{i=0}^{T-1}\sum_{j=0}^{T-1} c_i d_j x^{e_i+f_j}$$

Naive algorithm

Compute every $c_i d_j$, $0 \leq i, j < T$

- $O(T^2)$ coefficient multiplications and exponent additions
- Some coefficient additions, sorting, etc.

Less naive algorithms

- Good data structures [Johnson (1974), Yan (1998), Monagan-Pearce (2011)]
- Parallel implementation for efficiency [Monagan-Pearce (2009)]

Can we do better?

Sparsity of a product

$$(x^{14}+2x^7+2)\times(3x^{13}+5x^8+3) = 3x^{27}+5x^{22}+6x^{20}+10x^{15}+3x^{14}+6x^{13}+10x^8+6x^7+6$$

$\rightarrow$ 9 nonzero terms

$$(x^{14}+2x^7+2)\times(x^{14}-2x^7+2) = x^{28}+4$$

$\rightarrow$ 2 nonzero terms

- $f_\#$: *sparsity* (or number of nonzero terms) of a polynomial f

$$1 \leq (fg)_\# \leq f_\# g_\#$$

Difficulty

- Can we predict the output sparsity?
- Need of output-sensitive algorithms

Structural sparsity: a partial solution

$$(x^{14} + 2\,x^{7} + 2\,x^{0}) \times (x^{14} + -2\,x^{7} + 2\,x^{0}) = x^{28} + 0\,x^{21} + 0\,x^{14} + 0\,x^{7} + 4\,x^{0}$$

$\to$ *Structural sparsity* 5

Theorem [Arnold-Roche (2015)]

There is a randomized algorithm to compute the product of two sparse polynomials f, $g \in \mathbb{Z}[x]$ *of degree* $\leq D$ *and height* $\leq H$, *with bit cost quasi-linear in* $S \log D + T \log H$ *where* $T = \max(f_\#, g_\#, (fg)_\#)$ *and* S *is the structural sparsity of the product*

Structural sparsity: a partial solution

$$(x^{14} + 2\,x^{7} + 2\,x^{0}) \times (x^{14} + -2\,x^{7} + 2\,x^{0}) = x^{28} + 0\,x^{21} + 0\,x^{14} + 0\,x^{7} + 4\,x^{0}$$

$\rightarrow$ *Structural sparsity* 5

Theorem [Arnold-Roche (2015)]

There is a randomized algorithm to compute the product of two sparse polynomials f, $g \in \mathbb{Z}[x]$ *of degree* $\leq D$ *and height* $\leq H$, *with bit cost quasi-linear in* $S \log D + T \log H$ *where* $T = \max(f_\#, g_\#, (fg)_\#)$ *and* S *is the structural sparsity of the product*

Limitation

$$\left(\sum_{i=0}^{T-1} x^i\right) \times \left(\sum_{i=0}^{T-1} x^{Ti+1} - x^{Ti}\right) = X^{T^2} - 1 \text{ has structural sparsity } T^2 + 1$$

Division of sparse polynomials

Compute q and r such that $f = g \cdot q + r$ with $\deg r < \deg g$

$$x^{2n+1} - x^{2n} = (x^{n+1} - x^n + 1) \times x^n - x^n$$

$\to$ sparse output

$$x^{2n+1} + x^{2n} = (x^{n+1} - x^n + 1) \times (x^n + 2\,x^{n-1} + \cdots + 2) + x^n - 2\,x^{n-1} - \cdots - 2$$

$\to$ dense output

$$x^{2n+1} - x^{2n} = (x^{n+1} - 2\,x^n + 1) \times (x^n + x^{n-1} + 2\,x^{n-2} + \cdots + 2^{n-1})$$
$$+ (2^n - 3)x^n - x^{n-1} - 2\,x^{n-2} - \cdots - 2^{n-1}$$

$\to$ large coefficients

- Output size even more variable than in the case of multiplication!

Division algorithms

Classical Euclidean Algorithm

- Uses $O(g_\# q_\#)$ ring operations
- But many exponent comparisons $\to$ dominates the cost!
  - $O(f_\# + q_\# g_\#^2)$ using sorted lists
  - $O(f_\# + q_\# g_\# \log(f_\# + q_\# g_\#))$ using heap or geobucket [Johnson (1974), Yan (1998)]
- Space $\to$ heap size
  - $1 + q_\#$ [Johnson (1974)]
  - $O(g_\#)$ [Monagan-Pearce (2007)]

Best known algorithm

[Monagan-Pearce (2011)]

- Heap of size $\min(g_\#, q_\#)$
- Complexity $O(f_\# + q_\# g_\# \log \min(q_\#, g_\#))$

Output-sensitive but non-linear algorithm

GCD of sparse polynomials

$$\gcd(x^{ab}-1, x^{ab}-x^a-x^b+1) = x^{a+b-1}+x^{a+b-2}+\cdots+x^a-x^{b-1}-x^{b-2}-\cdots-1$$

[Schinzel (2002)]

Hardness results

- *Testing if two sparse polynomials over $\mathbb{Z}$ are coprime is* coNP*-hard* [Plaisted (1984)]
- Generalization over $\mathbb{F}_q$

  [von zur Gathen-Karpinski-Shparlinski (1996), Karpinski-Shparlinski (1999), Kaltofen-Koiran (2005)]

Upper bounds

Let $f, g \in \mathbb{Z}[x]$ with a *fixed sparsity and height*:

- If f or g is *cyclotomic-free*, $\gcd(f, g)$ in complexity $\tilde{O}(\log D)$

  [Filaseta-Granville-Schinzel (2008)]
- Compute a polynomial with *the same roots as* $\gcd(f, g)$ *in* $\bar{\mathbb{Q}}$ in complexity $\tilde{O}(\log D)$ (in particular : coprimality test) [Amoroso-Leroux-Sombra (2015)]

Are there output-sensitive algorithms that computes GCD and the Bézout coefficients?

Challenges for sparse polynomial operations

Output sensitivity

- Output size is usually **not** determined by input size
- Algorithms must be output sensitive
- Depending on operations, output size may be very variable

Known complexities

- Multiplication and division: output-sensitive quadratic algorithms
- GCD: exponential time in the general case

Today: Output-sensitive quasi-linear algorithms for multiplication and exact division

- Ignored: Divisibility testing

1. Known algorithms and challenges

2. The main tool: sparse interpolation

3. Fast multiplication and exact division algorithms

Definition of the problem

Input: A sparse polynomial $f \in R[x]$ in an *implicit representation*
Bounds on D, H and/or T

Output: The sparse representation of f

Implicit representations

- Straight-line program (SLP), *a.k.a* arithmetic circuit
- Blackbox $\rightarrow$ evaluation program for f on elements of R
- Extended blackbox $\rightarrow$ evaluate f outside R
  - Modular blackbox: if $R = \mathbb{Z}$, evaluate $f(\theta) \bmod m$ for any $m \in \mathbb{Z}$
  - Remainder blackbox: evaluate on $\theta \in R[x]/\langle g \rangle$ for any $g \in R[x]$

Remark

- Given an SLP, one can compute explicitely f
- Infeasible because of intermediate expression swell

Many variants of the problem

Ring of coefficients

- $\mathbb{Z}$ or $\mathbb{Q}$: size growth $\to$ modular techniques
- *Large* finite fields
- Finite fields of *large characteristic*
- Small finite fields

Input representation

- Blackboxes: count the number of *queries* + extra arithmetic / bit operations
- SLP: count the cost of each *query*
  - arithmetic cost: number L of instructions
  - bit cost: $\tilde{O}(L \log H)$ where H bounds the height of the constants

Randomization

- Deterministic
- Monte Carlo randomization
- Las Vegas randomization

Blackbox algorithm using geometric progressions

$$f = \sum_{i=0}^{T-1} c_i x^{e_i} \to \begin{pmatrix} f(1) \\ f(\omega) \\ \vdots \\ f(\omega^n) \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 1 \\ \omega^{e_0} & \cdots & \omega^{e_{t-1}} \\ \vdots & & \vdots \\ \omega^{ne_0} & \cdots & \omega^{ne_{t-1}} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{T-1} \end{pmatrix}$$

Property

- The sequence $(f(\omega^j))_{j\geq 0}$ is linearly recurrent, of minimal polynomial $\prod_i (x - \omega^{e_i})$

Algorithm

[Prony (1795), Ben-Or–Tiwari (1988), …]

1. Evaluate f at $1, \omega, \ldots, \omega^{2T-1}$
2. Compute the minimal polynomial of $(f(\omega^j))_j$ using Berlekamp-Massey algorithm
3. Compute its roots and obtain the exponents $e_0, \ldots, e_{T-1}$
4. Solve a transposed Vandermonde system to get the coefficients $c_0, \ldots, c_{T-1}$

SLP algorithm using cyclic extensions

Compute explicitly $f \bmod x^p - 1 = \sum_i c_i x^{e_i \bmod p}$ for some prime p [Garg-Schost (2009)]

Loss of information

- Exponents known only *modulo p*
- Possible *collisions* between monomials

Reconstruction of full exponents

- Use several p_j's and (polynomial) Chinese remaindering, *diversification*, …

  [Garg-Schost (2009), Giesbrecht-Roche (2011), …]
- Encode exponents into coefficients *à la* Paillier or using derivatives

  [Arnold-Roche (2015), Huang (2019)]

Deal with collisions

- Large enough prime and/or many primes to avoid any collision [Garg-Schost (2009)]
- Accept collisions and reconstruct f iteratively

  [Arnold-Giesbrecht-Roche (2013), Huang (2019)]

Summary of main algorithms

Blackbox algorithms, using geometric progressions

- Require $O(T)$ queries to the blackbox
- Arithmetic complexity : $\mathrm{poly}(T, D)$
- Remark: over $\mathbb{Z}$, $f(\omega^j)$ has bit size $\Omega(D)$

SLP algorithms, using cyclic extensions

- Over $\mathbb{Z}$ and $\mathbb{F}_q$, bit cost $\mathrm{poly}(T, \log D, \log H)$
- Best known complexity: $\tilde{O}(T \log D \log q)$ over $\mathbb{F}_q$ if $q = \tilde{\Omega}(DT)$ [Huang (2019)]

New algorithm

Input: A *modular blackbox* for $f \in \mathbb{Z}[x]$, bounds on T, D and H
Complexity: $\tilde{O}(T(\log D + \log H))$
General idea: Combine both techniques

First ingredient: compute exponents of $f \bmod x^p - 1$

Choice of p

- Prime such that $(f \bmod x^p - 1)_\# \geq \frac{5}{6} f_\#$ w.h.p.
- Random choice of $p = O(T \log D)$

Evaluations in a small field $\mathbb{F}_q$

- If ω is a p-PRU in $\mathbb{F}_q$, $f(\omega^j) = (f \bmod x^p - 1)(\omega^j)$
- Small q for efficiency reasons
- Coefficients should remain nonzero modulo $q \to q = \text{poly}(T \log H)$

Algorithm

1. Compute a p-PRU $\omega \in \mathbb{F}_q$ — Effective Dirichlet's theorem
2. Evaluate f at $1, \omega, \ldots, \omega^{2T-1}$ — $2T$ queries
3. Compute the minimal polynomial of $(f(\omega^j))_j$ — $\tilde{O}(T \log q)$
4. Compute its roots and get the exponents by Bluestein's transform — $\tilde{O}(p \log q)$

Second ingredient: compute $f \bmod x^p - 1$

Evaluations in a larger ring

- $\mathbb{F}_q$ is too small $\rightarrow$ coefficients known modulo q
- Use larger ring where coefficients can be represented
- Using large finite field is too costly (primality testing, etc.)

$\rightarrow$ Ring $\mathbb{Z}/q^k\mathbb{Z}$ where $q^k > 2H$

Transposed Vandermonde system solving modulo q^k

- Evaluation on a p-PRU $\omega_k \in \mathbb{Z}/q^k\mathbb{Z}$
- Invertible matrix $\rightarrow$ ω_k should be *principal*
- Transposed algorithm of fast (dense) interpolation

Algorithm

1. Compute a p-PRU $\omega_k \in \mathbb{Z}/q^k\mathbb{Z}$ from $\omega \in \mathbb{F}_q$ — Newton iteration
2. Evaluate f at $1, \omega_k, \dots, \omega_k^{T-1}$ — T queries
3. Solve a transposed Vandermonde system, build using the exponents — $\tilde{O}(Tk \log q)$

Third ingredient: Embed exponents into coefficients

Compute both $f(x)$ and $f((1+q^k)x)$ modulo $\langle x^p - 1, q^{2k} \rangle$

Paillier-like embedding

- $(1+q^k)^{e_i} = 1 + e_i q^k \mod q^{2k}$
- Image of a monomial $c_i x^{e_i}$ of f:
  - in $f(x) \bmod \langle x^p - 1, q^{2k} \rangle \to c_i x^{e_i \bmod p}$
  - in $f((1+q^k)x) \bmod \langle x^p - 1, q^{2k} \rangle \to c_i(1 + e_i q^k) x^{e_i \bmod p}$

Collisions

- If $c_i x^{e_i}$ is collision-free modulo $x^p - 1 \to$ reconstruct both c_i and e_i
- Possibly noisy terms from collisions $e_i = e_j \bmod p$

$\to$ Compute f^* such that $(f - f^*)_\# \leq \frac{1}{2} f_\#$ w.h.p.

Complete algorithm

Algorithm

1. $f^* \leftarrow 0$
2. Repeat $\log T$ times :
3. Compute $p, q, \omega \in \mathbb{F}_q, \omega_k \in \mathbb{Z}/q^{2k}\mathbb{Z}$
4. Compute exponents of $(f - f^*) \bmod \langle x^p - 1, q \rangle$ — First ingredient
5. Compute $(f - f^*) \bmod \langle x^p - 1, q^{2k} \rangle$ — Second ingredient
6. Compute $(f - f^*)((1 + q^k)x) \bmod \langle x^p - 1, q^{2k} \rangle$ — Second ingredient
7. Reconstruct collision-free monomials plus some noise — Third ingredient
8. Update f^*
9. Return f^*

Theorem

[Giorgi-G.-Perret du Cray-Roche (2022)]

Given a modular blackbox or an SLP for $f \in \mathbb{Z}[x]$, the algorithm returns the sparse representation of f with probability $\geq \frac{2}{3}$, and has bit complexity $\tilde{O}(T(\log D + \log H))$

1. Known algorithms and challenges

2. The main tool: sparse interpolation

3. Fast multiplication and exact division algorithms

How to use sparse interpolation?

Multiplication

- Given $f, g \in \mathbb{Z}[x]$, compute $f \times g$
- Desired complexity: $\tilde{O}(\max(f_\#, g_\#, (fg)_\#)(\log D + \log H))$
- $f \times g$ is a special SLP $\to$ use sparse interpolation
- Caveats:
  - No a priori bound on $(fg)_\#$
  - Cost of sparse polynomial evaluation

Exact division

- Given $f, g \in \mathbb{Z}[x]$ s.t $g \mid f$, compute f/g
- Desired complexity: $\tilde{O}(\max(f_\#, g_\#, (f/g)_\#)(\log D + \log H))$
- Caveats:
  - No bound on $(f/g)_\#$, nor on $\|f/g\|_\infty$
  - f/g is a special SLP *with divisions*

Evaluation of sparse polynomials

Given $f = \sum_{i=0}^{T-1} c_i x^{e_i}$ and α, compute $f(\alpha)$

Not polynomial-time over $\mathbb{Z}$

- If $\alpha \neq \pm 1$, α^D needs D bits to be written

Over a finite field $\mathbb{F}_q$

- α^e: $O(\log e)$ operations in $\mathbb{F}_q$
- Computation of $f(\alpha)$:
  - $O(T \log D)$ operations in $\mathbb{F}_q$
  - $\tilde{O}(T \log D \log q)$ bit operations — not quasi-linear!
- Slight improvement: $O(\log D + T \log D / \log\log D)$ or $O(T \log D / \log T)$ [Yao (1976)]

Avoid bound on T

Standard doubling strategy

- Try to compute $f \times g$ or f/g with bound T
- If the result is incorrect, double T

$\rightarrow$ Requires a *fast* equality test for $f \times g = h$

Sparse product verification

[Giorgi-G.-Perret du Cray (2019, 2022)]

- Classical method: evaluate f, g, h at a random point $\rightarrow$ too costly
- Main idea: evaluate $(f \times g) \bmod x^p - 1$ at a random point, for a random p
  - Sparse and structured vector-matrix-vector product
- Bit complexity $\tilde{O}(T(\log D + \log H))$

Overview of multiplication algorithm

Two-level algorithm

1. Reduce f, g and their derivatives f', g' modulo $x^p - 1$
2. Use sparse interpolation and verification to compute:
   - $(f \bmod x^p - 1) \times (g \bmod x^p - 1)$
   - $(f' \bmod x^p - 1) \times (g \bmod x^p - 1)$ and $(f \bmod x^p - 1) \times (g' \bmod x^p - 1)$
3. Deduce $(fg) \bmod x^p - 1$ and $(fg)' \bmod x^p - 1$ and reconstruct fg

Choice of p and complexity

- We want no collision w.h.p. $\rightarrow p = O(f_\#^2 g_\#^2 \log D)$ $\qquad (fg)_\# \leq f_\# g_\#$
- Reduction and reconstruction : $\tilde{O}(T(\log D + \log H))$ $\qquad T = \max(f_\#, g_\#, (fg)_\#)$
- Sparse interpolation of polynomials of degree $O(p) \rightarrow \text{poly}(\log D)$ is good enough

Theorem

[Giorgi-G.-Perret du Cray (2019)]

Randomized algorithm of complexity $\tilde{O}(T(\log D + \log H))$ for sparse polynomial product over $\mathbb{Z}$ or $\mathbb{F}_q$ of large characteristic

New difficulties for an exact division algorithm

Evaluation

- To compute $(f/g) \bmod x^p - 1$, g must be coprime with $x^p - 1$
- To work modulo some q, they must remain coprime in $\mathbb{F}_q[x]$

$\rightarrow$ Additional conditions on p and q

No two-level approach

- Two-level algorithm for $f \times g$:
  - Compute $(f \bmod x^p - 1) \times (g \bmod x^p - 1)$
  - Reduce the result to get $(fg) \bmod x^p - 1$
- There is no reason for $(g \bmod x^p - 1)$ to divide $(f \bmod x^p - 1)$

$\rightarrow$ Requires a truly efficient sparse interpolation algorithm

Bounds on T and H

- $(f/g)_\#$ can be as large as D
- Height of f/g can be as large as $H^{O((f/g)_\#)}$
- Both bounds must be discovered at the same time

$\rightarrow$ Modular product verification algorithm [Giorgi-G.-Perret du Cray (2022)]

Exact division algorithm

Algorithm

1. $h \leftarrow 0$; $T \leftarrow g_\#$; $H \leftarrow$ height of g
2. While $T \geq 1$:
3. Compute $h_p \leftarrow (f/g - h) \bmod \langle x^p - 1, q^{2k} \rangle$ using sparse interpolation
4. If $f = g \times (h + h_p) \mod x^p - 1$: *modular verification alg.*
5. Compute new terms of h from h_p
6. $T \leftarrow T/2$
7. Else: $H \leftarrow H^2$
8. If $f = g \times h$: return h *sparse verification alg..*
9. Else: restart, with T twice as large

Theorem

[Giorgi-G.-Perret du Cray-Roche (2022)]

Given $f, g \in \mathbb{Z}[x]$ such that $g \mid f$, the algorithm returns f/g with probability $\geq \frac{2}{3}$, and has bit complexity $\tilde{O}(T(\log D + \log H))$

Conclusion

Overlooked in this presentation

Divisibility testing

- Given f and g, does g divides f?
- Easy if $\deg(g)$ or $\deg(f) - \deg(g)$ is small
- New polynomial-time algorithm in some special cases [Giorgi-G.-Perret du Cray (2021)]

$\rightarrow$ Problem not known to be polynomial in general

Sparse interpolation without sparsity bound

- *Early termination* techniques [Kaltofen-Lee (2003)]
- Quasi-linear running time?
  - Classical method $\rightarrow$ requires a large prime number
  - D5-like approach to avoid primality testing [Giorgi-G.-Perret du Cray-Roche (2022)]
- Non suitable for multiplication and exact division $\rightarrow$ too costly

Conclusion and open questions

Multiplication and exact division

- Multiplication of sparse polynomials
  - First quasi-linear algorithm over $\mathbb{Z}$ or $\mathbb{F}_q$ with large characteristic
- Exact division of sparse polynomials
  - First quasi-linear algorithm over $\mathbb{Z}$
  - First "quasi-linear in T" algorithm over $\mathbb{F}_q$ with large characteristic

Sparse interpolation

- First quasi-linear algorithm over $\mathbb{Z}$

Open questions

- Quasi-linear algorithms over $\mathbb{F}_q$ with small characteristic many cancellations
- Division with remainder, remainder only
- GCD with Bézout coefficients

Conclusion and open questions

Multiplication and exact division

- Multiplication of sparse polynomials
  - First quasi-linear algorithm over $\mathbb{Z}$ or $\mathbb{F}_q$ with large characteristic
- Exact division of sparse polynomials
  - First quasi-linear algorithm over $\mathbb{Z}$
  - First "quasi-linear in T" algorithm over $\mathbb{F}_q$ with large characteristic

Sparse interpolation

- First quasi-linear algorithm over $\mathbb{Z}$

Open questions

- Quasi-linear algorithms over $\mathbb{F}_q$ with small characteristic many cancellations
- Division with remainder, remainder only
- GCD with Bézout coefficients

Thank you!