Sparse interpolation over the integers

Bruno Grenet

Joint work with P. Giorgi, A. Perret du Cray and D. S. Roche

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES - INFORMATIQUE

UGA Université Grenoble Alpes

Dagstuhl seminar

From Sparse Interpolation to Signal Processing: New Synergies

July 11., 2025

General definition of the problem

Sparse interpolation

Inputs: A way to *evaluate* a sparse polynomial $f \in \mathrm{R}[x]$
Bounds $\delta \geq f^\circ$, $\tau \geq f_\#$, and $\gamma \geq f_\infty$ (optional)

Output: The sparse representation of $f = \sum_{i=0}^{t-1} c_i x^{e_i}$, $c_i \in \mathrm{R}_{\neq 0}$

Notations

- R: ring of coefficients
- x: variable, or tuple of variables
- f°: degree of f
- $f_\#$: *sparsity* of f, that is number of non-zero terms
- f_∞: *height* of f $\simeq$ measure of the size of the coefficients (if this makes sense)

Many variants

Ring of coefficients

- $\mathbb{Z}$ or $\mathbb{Q}$ — size growth $\rightarrow$ modular techniques
- $\mathbb{R}$ or $\mathbb{C}$ — precision issues
- Finite fields — large/small size/characteristic
- Modular rings — zero divisors

Number of variables

- Univariate polynomials
- Multivariate polynomials — Kronecker substitution $\rightarrow$ univariate case

Input representation

- Fixed evaluations
- Black box
- Arithmetic circuit, *a.k.a* Straight-Line Program (SLP)

Many variants

Ring of coefficients

- $\mathbb{Z}$ or $\mathbb{Q}$ — size growth $\rightarrow$ modular techniques
- $\mathbb{R}$ or $\mathbb{C}$ — precision issues
- Finite fields — large/small size/characteristic
- Modular rings — zero divisors

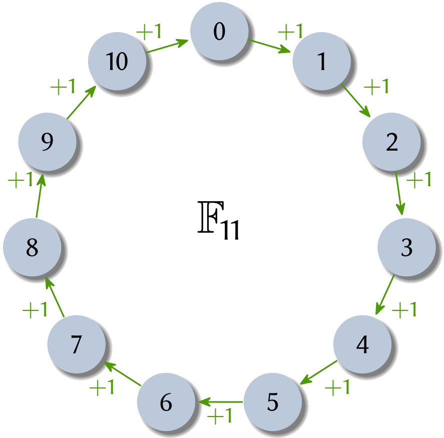
Number of variables

- Univariate polynomials
- Multivariate polynomials — Kronecker substitution $\rightarrow$ univariate case

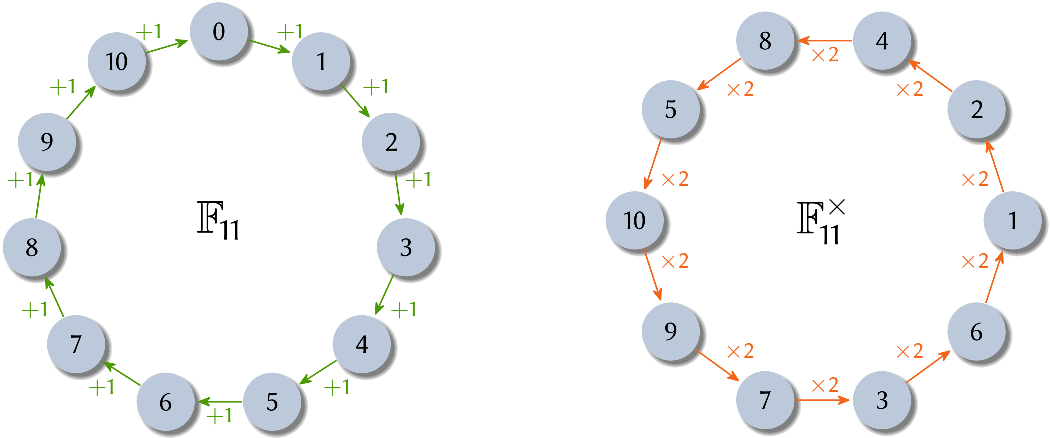
Input representation

- Fixed evaluations
- Black box
- Arithmetic circuit, *a.k.a* Straight-Line Program (SLP)

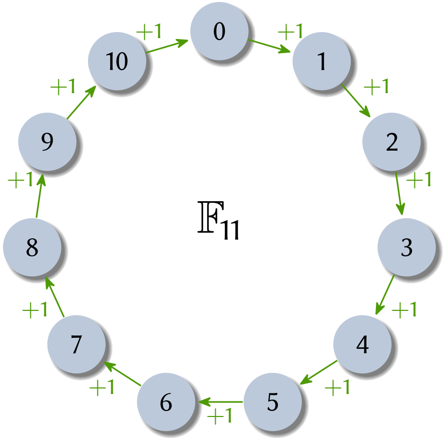
A primer on (prime) finite fields

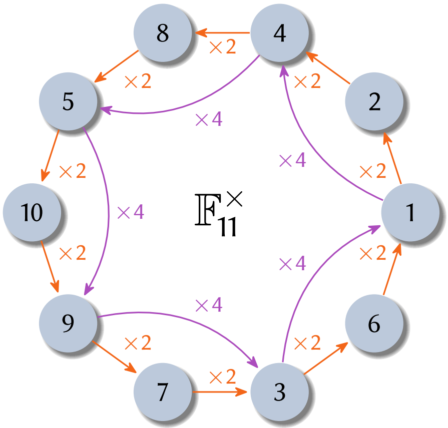


A primer on (prime) finite fields

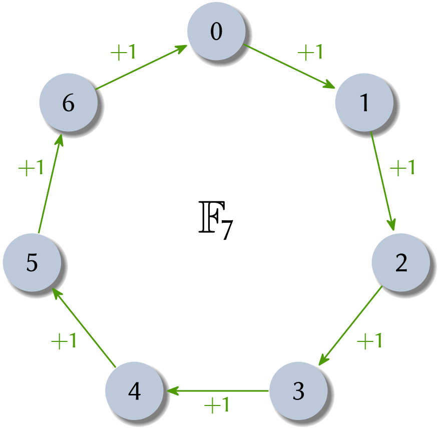


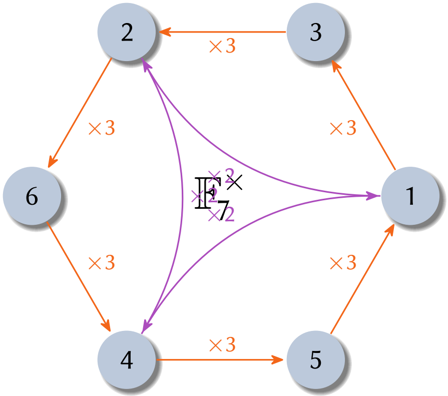
A primer on (prime) finite fields



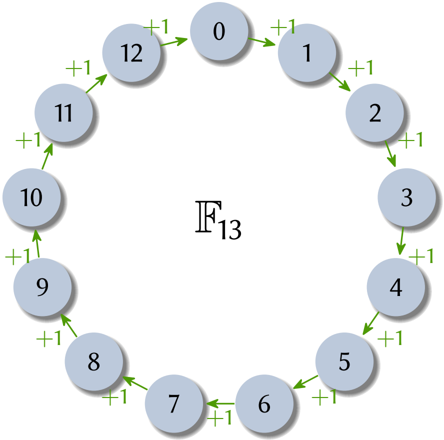


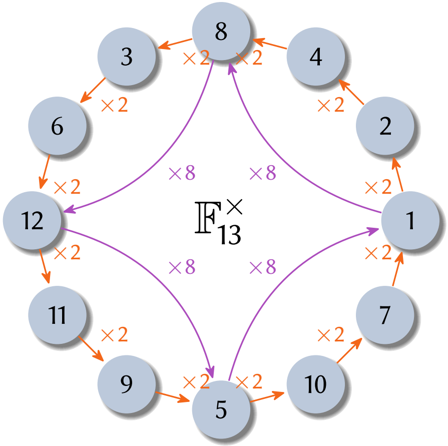
A primer on (prime) finite fields



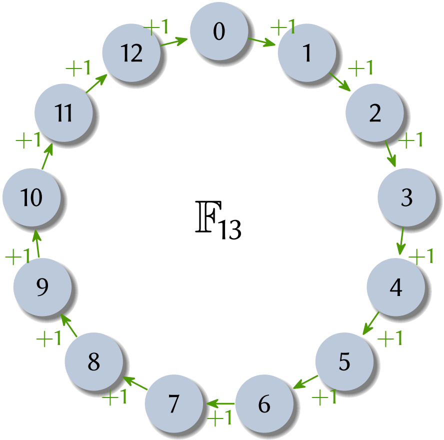


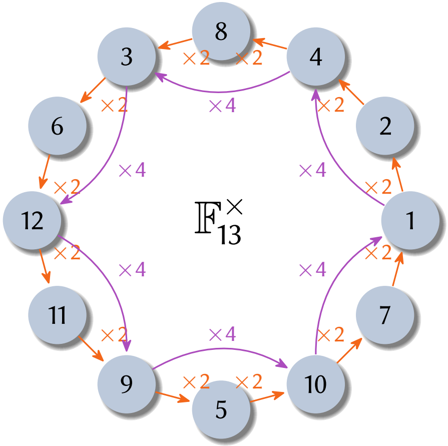
A primer on (prime) finite fields



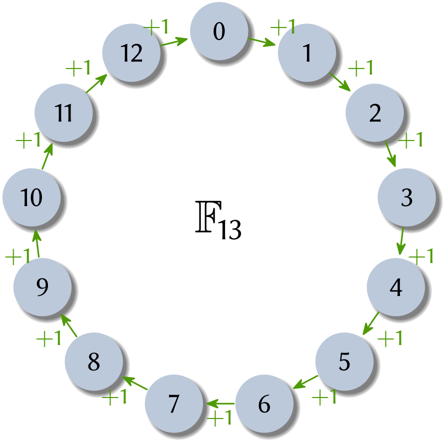


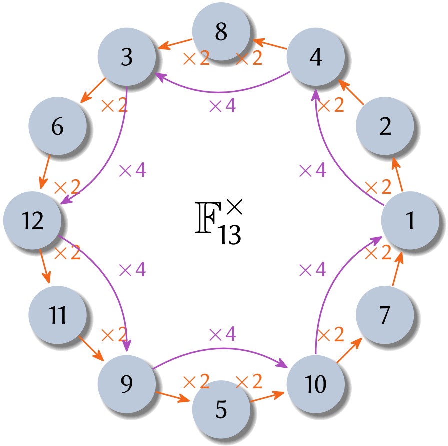
A primer on (prime) finite fields





A primer on (prime) finite fields





- All four operations $+$, $-$, $\times$ and $/$ are well-defined in $\mathbb{F}_q$
- Each $\alpha \neq 0$ is a *root of unity* of some order k that divides $q - 1$ $\qquad \alpha^k = 1$

Contents

Contents

Algorithm *à la* Prony / Ben-Or–Tiwari

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

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

Theorem

[Blahut (1979)]

If ω has order $> f^\circ$, the minimal polynomial of $(f(\omega^j))_{j\geq 0}$ is $\Lambda(x) = \prod_{i=0}^{t-1}(x - \omega^{e_i})$.

Algorithm

1. Evaluate f at $1, \omega, \ldots, \omega^{2\tau-1}$ — *black box*
2. Compute the minimal polynomial Λ of $(f(\omega^i))_i$ — *Prony polynomial*
3. Compute the roots $\rho_0, \ldots, \rho_{t-1}$ of Λ
4. Compute their discrete logarithms $e_0, \ldots, e_{t-1}$ — $\rho_i = \omega^{e_i}$
5. Compute $c_0, \ldots, c_{t-1}$ by transposed Vandermonde system solving

Complexity analysis over $\mathbb{F}_q$: fast steps

Minimal polynomial computation

Given $(f(\omega^i))_{0 \le i < 2\tau}$, compute its minimal polynomial

- LFSR synthesis, error correcting codes [Berlekamp (1968), Massey (1969)]
- Padé approximant, Euclid algorithm [Brent-Gustavson-Yun (1980)]
- Hankel system solving [Lanczos (1952)]

$\tilde{\mathcal{O}}(\tau)$ operations in $\mathbb{F}_q = \tilde{\mathcal{O}}(t \log q)$ bit operations

Coefficients computation

Given $(f(\omega^i))_{0 \le i < t}$ and $(\omega^{e_i})_{0 \le i < t}$, compute $c_0, \ldots, c_{t-1}$

- Vandermonde system solving $\Leftrightarrow$ (dense) interpolation [Borodin-Moenck (1974)]
- Transposed Vandermonde syst. solv. [Kaltofen-Lakshman (1992), Bostan-Lecerf-Schost (2003)]

$\tilde{\mathcal{O}}(t)$ operations in $\mathbb{F}_q = \tilde{\mathcal{O}}(t \log q)$ bit operations

Complexity analysis over $\mathbb{F}_q$: not-so-fast steps

Root finding

Given $\Lambda = \sum_{i=0}^{t-1} \lambda_i x^i$, compute its t *non-zero distinct* roots $\rho_0, \ldots, \rho_{t-1} \in \mathbb{F}_q$

- $\Gamma \leftarrow \textsc{gcd}\left(\Lambda, (x+\alpha)^{\frac{q-1}{2}}\right)$ for random α [Rabin (1980)]
- Recursion with Γ and Λ/Γ [Berlekamp (1970)]

$\tilde{\mathcal{O}}(t\log(q))$ operations in $\mathbb{F}_q = \tilde{\mathcal{O}}(t\log^2 q)$ bit operations

Discrete logarithms

Given $\rho_0, \ldots, \rho_{t-1}$, compute $e_i \in [0, \ldots, \delta]$ s.t. $\rho_i = \omega^{e_i}$ for $0 \le i < t$

- *Baby steps/giant steps* algorithm [Shanks (1971)]
- Use bound δ and combine t computations [Pollard (1978), Kuhn-Struik (2001)]

$O(\sqrt{\delta t})$ operations in $\mathbb{F}_q = \tilde{\mathcal{O}}(\sqrt{\delta t}\log q)$ bit operations

Conclusion on Prony / Ben-Or–Tiwari algorithm

Theorem

Given black box access to $f \in \mathbb{F}_q[x]$ and bounds $\tau \geq f_\#$ and $\delta \geq f^\circ$, one can compute the sparse representation of f in $\tilde{\mathcal{O}}(\sqrt{\tau\delta}\log q + \tau\log^2 q)$ bit operations

Good and bad news

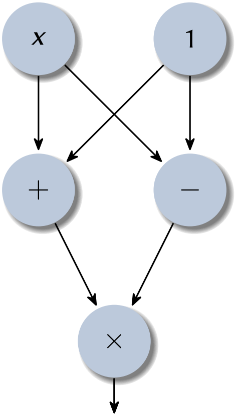
- Quasi-linear in τ, linear (optimal) number of evaluations
- Bound $\tau \geq f_\#$ not required $\rightarrow$ *early termination* [Kaltofen-Lee (2003)]
- Polynomial in δ, rather than $\log\delta$ $\rightarrow$ not polynomial in the output size

Other rings

- $\mathbb{Z}/\mathbb{Q}$:
  - large evaluations (bit size $O(\delta)$) [Ben-Or–Tiwari (1988)]
  - Compute modulo p where $p-1$ is smooth $\rightarrow$ fast discrete log. [Kaltofen (1988/2010)]
- Modular rings: works as long as ω is a *principal* root of unity of large order

Contents

Arithmetic circuit / Straight-line program



Input: $r_0 := x$
Constant: $r_1 := 1$
2. $r_2 := r_0 + r_1$
3. $r_3 := r_0 - r_1$
4. $r_4 := r_2 \times r_3$

$$f(x) = x^2 - 1$$

Given an SLP for $f \in R[x]$ (and bounds), compute its sparse representation

Folding the polynomials

- From an SLP, f can be computed explicitly in time $O(\delta)$ *expression swell*
- Compute $f \bmod x^p - 1 = \sum_i c_i x^{e_i \bmod p}$ for some prime p [Garg-Schost (2009)]
  - Exponents known only *modulo p*
  - Possible *collisions* between monomials

Exponent embedding

- If $f = \sum_i c_i x^{e_i}$, $xf' = \sum_i c_i e_i x^{e_i}$ [Huang (2019)]
  - requires $\mathrm{char}(\mathbb{F}_q) \geq f^\circ$
- $f((1+q)x) = \sum_i c_i(1+e_i q)x^{e_i} = f + q \cdot (xf')$ in $\mathbb{Z}/q^2\mathbb{Z}[x]$ [Arnold–Roche (2015)]

Deal with collisions

- Avoid collisions with random primes or many primes [Garg-Schost (2009)]
  - If $p = \Omega(\tau^2 \log \delta)$ is random, no collision *w.h.p.*
- Accept some collisions and correct errors [Arnold-Giesbrecht-Roche (2013), Huang (2019)]
  - If $p = \Omega(\tau \log \delta)$ is random, $\leq \frac{1}{6}\tau$ collisions *w.h.p*
  - Compute an *approximation* f_* such that $(f - f_*)_\# \leq \frac{1}{2} f_\#$ *w.h.p.*

Algorithm *à la* Garg–Schost

[Garg-Schost (2009), Huang (2019)]

Algorithm

Inputs: SLP for $f \in \mathbb{F}_q[x]$, $\mathrm{char}(\mathbb{F}_q) \geq f^\circ$, and bounds $\tau \geq f_\#$, $\delta \geq f^\circ$

1. $f_* \leftarrow 0$
2. Repeat $\log(\tau)$ times:
3. $\quad p \leftarrow$ random prime in $[\lambda, 2\lambda]$ for $\lambda = \tilde{\mathcal{O}}(\tau \log \delta)$
4. $\quad f_p^{(0)} \leftarrow f \bmod x^p - 1$
5. $\quad f_p^{(1)} \leftarrow (xf') \bmod x^p - 1$ *from* $f((1+q)x) \bmod x^p - 1$
6. $\quad$ For each pair $cx^d \in f_p^{(0)}$, $c'x^d \in f_p^{(1)}$: add $c \cdot x^{c'/c}$ to f_* *if* $c'/c \in \{0, \ldots, \delta\}$
7. Return f_*

Complexity analysis

- $O(\log \tau)$ *probes* $= \tilde{\mathcal{O}}(sp \log \tau)$ operations in $\mathbb{F}_q = \tilde{\mathcal{O}}(s\tau \log \delta \log q)$ bit operations

s: SLP size

Remarks on Garg–Schost algorithm

Huang's variant is almost quasi-linear!

- Output size: $O(\tau(\log\delta + \log q))$, complexity: $\tilde{\mathcal{O}}(\tau \log\delta \log q)$
- Hard to avoid: *probing* the circuit is already non-quasi-linear

Other base rings

- Smaller characteristic
  - No exponent embedding anymore
  - Several techniques, such as *diversification*
  - Best complexity: $O(s\tau \log^2\delta(\log\delta + \log q))$ [Arnold-Giesbrecht-Roche (2014)]
- Over the integers
  - Coefficient growth $\to$ modular techniques
  - Best complexity: $O(s\tau \log^3\delta \log\gamma)$ where $\gamma \geq f_\infty$ [Perret du Cray (2023)]

Contents

Result

Inputs: *Modular black box* for $f \in \mathbb{Z}[x]$
Bounds $\tau \geq f_{\#}, \delta \geq f^{\circ}, \gamma \geq f_{\infty}$
Complexity: $\tilde{\mathcal{O}}(\tau(\log \delta + \log \gamma))$ bit operations

Modular black box

- Given α and m, compute $f(\alpha) \bmod m$
- Can be implemented given an arithmetic circuit / SLP
- Pure black box: evaluations on $\mathbb{Z} \setminus \{0, \pm 1\}$ have size $\Omega(\delta)$

The strategy

- General structure: *à la* Garg-Schost
- Computing $f \bmod x^p - 1$: *à la* Prony / Ben-Or–Tiwari
- Work over several rings of different sizes to make it efficient

Details on the strategy

1. Compute the exponents of $f \bmod x^p - 1$
   - Work in a small field $\mathbb{F}_q \to$ no coefficient should vanish modulo q

2. Compute the coefficients of $f \bmod x^p - 1$
   - Work in a larger *ring* $\mathbb{Z}/q^k\mathbb{Z} \to$ the coefficients must be exactly representable

3. Compute the (non-colliding) exponents of f
   - Embed the exponents into the coefficients $\to$ work with both $f(x)$ and $f((1+q^k)x)$

4. Recurse
   - Steps 1-3 compute an *approximation* f_* of f
   - Restart with f replaced by $(f - f_*)$

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

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

- If ω has order p in $\mathbb{F}_q$, $f(\omega^j) = (f \bmod x^p - 1)(\omega^j)$
- Small random q for efficiency reasons
  - Only require coefficients to be nonzero mod q — $q = \mathrm{poly}(\tau \log \gamma)$
- Random p to prevent too many collisions — $p = O(\tau \log \delta)$
  - p must divide $q - 1$ to have $\omega \in \mathbb{F}_q^\times$ of order p

Algorithm: first part of Prony's method

Inputs: f and $\omega \in \mathbb{F}_q^\times$ of order p — *to be computed*

1. Evaluate f at $1, \omega, \dots, \omega^{2\tau-1}$ — 2τ queries
2. Compute the minimal polynomial of $(f(\omega^j))_j$ — $\tilde{\mathcal{O}}(\tau \log q)$
3. Compute its roots and get the exponents — $\tilde{\mathcal{O}}(p \log q)$

Complexity analysis

$$\tilde{\mathcal{O}}(\tau \log q + p \log q) = \tilde{\mathcal{O}}(\tau \log \delta \log \log \gamma)$$

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

Evaluations in a larger ring

- $\mathbb{F}_q$ is too small $\to$ coefficients known modulo q
  - Use larger ring where coefficients can be represented
  - Using large finite field is too costly (primality/irreducibility testing)
- Ring $\mathbb{Z}/q^k\mathbb{Z}$ where $q^k > 2\gamma$ $\quad$ $k = O(\log\gamma/\log q)$

Algorithm: second part of Prony's method

Inputs: $\boxed{f}$ and $\omega_k \in (\mathbb{Z}/q^k\mathbb{Z})^\times$ of order p $\quad$ *to be computed*

the exponents $e_0, \dots, e_{t-1}$ of $f \bmod x^p - 1$

1. Evaluate $\boxed{f}$ at $1, \omega_k, \dots, \omega_k^{t-1}$ $\quad$ τ queries
2. Solve the system $\begin{pmatrix} 1 & \cdots & 1 \\ \omega_k^{e_0} & \cdots & \omega_k^{e_{t-1}} \\ \vdots & \ddots & \vdots \\ \omega_k^{(t-1)e_0} & \cdots & \omega_k^{(t-1)e_{t-1}} \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{t-1} \end{pmatrix} = \begin{pmatrix} f(1) \\ f(\omega_k) \\ \vdots \\ f(\omega_k^{t-1}) \end{pmatrix}$ $\quad$ $\tilde{\mathcal{O}}(\tau k \log q)$

Complexity analysis

$$\tilde{\mathcal{O}}(\tau k \log q) = \tilde{\mathcal{O}}(\tau \log\gamma)$$

Third ingredient: Embed exponents into coefficients

An even slightly larger ring

- Consider $f((1+q^k)x)$ in $\mathbb{Z}/q^{2k}\mathbb{Z}[x]$:

$$f((1+q^k)x) \bmod x^p - 1 = (f(x) + q^k \cdot (xf')(x)) \bmod x^p - 1$$

Modified algorithm

Inputs f and $\omega_{2k} \in (\mathbb{Z}/q^{2k}\mathbb{Z})^\times$ of order p — *to be computed* the exponents $e_0, \ldots, e_{t-1}$ of $f \bmod x^p - 1$

1. Evaluate f at $1, \omega_{2k}, \ldots, \omega_{2k}^{t-1}$ — τ queries
2. Evaluate f at $1+q^k, (1+q^k)\omega_{2k}, \ldots, (1+q^k)\omega_k^{t-1}$ — τ queries
   - Obtain evaluations $(xf')(1), (xf')(\omega_{2k}), \ldots, (xf')(\omega_{2k}^{t-1})$ — $\tilde{\mathcal{O}}(\tau k \log q)$
3. Solve the two transposed Vandermonde systems — $\tilde{\mathcal{O}}(\tau k \log q)$
   - Get *non-colliding* terms of f

Complexity analysis

$$\tilde{\mathcal{O}}(\tau k \log q) = \tilde{\mathcal{O}}(\tau \log \gamma)$$

Fourth ingredient: $\omega \in \mathbb{F}_q$ and $\omega_{2k} \in \mathbb{Z}/q^{2k}\mathbb{Z}$ of multiplicative order p

Compute p, q, ω and ω_{2k} together

- p must divide $q-1$: take $q = kp+1$ for some k — effective Dirichlet theorem
- ω has order $p \quad \Leftrightarrow \quad \omega = \alpha^{(q-1)/p} \neq 1$ for some α
- ω_{2i} has order p modulo $q^{2i} \Rightarrow \omega_{2i}$ has order p modulo q^i — Hensel lifting

Algorithm

1. Sample a random prime $p \in [\lambda, 2\lambda]$ with $\lambda = O(\tau \log \delta)$ — $O(\mathrm{poly}(\log \lambda))$
2. Sample a random prime $q \in \{kp+1 : 1 \leq k \leq \lambda^5\}$ — $O(\mathrm{poly}(\log \lambda))$
3. Sample a random $\alpha \in \mathbb{F}_q$ until $\omega = \alpha^{(q-1)/p} \neq 1$ — $O(\mathrm{poly}(\log q/p))$
4. Lift ω to $\omega_{2k} \in \mathbb{Z}/q^{2k}\mathbb{Z}$ of same order — $\tilde{\mathcal{O}}(k \log p \log q)$
5. Return $(p, q, \omega, \omega_{2k})$

Complexity analysis

$$\tilde{\mathcal{O}}(\mathrm{poly}(\log \lambda) + k \log p \log q) = \tilde{\mathcal{O}}(\mathrm{poly}(\log(\tau \log \delta)) + \log(\gamma) \log(\tau \log \delta))$$

Add recursion: full algorithm

Algorithm

1. $f_* \leftarrow 0$
2. Repeat $\log \tau$ times :
3. Compute p, q, $\omega \in \mathbb{F}_q$, $\omega_{2k} \in \mathbb{Z}/q^{2k}\mathbb{Z}$ — Fourth ingredient
4. Compute the exponents of $(f - f_*) \bmod x^p - 1$ in $\mathbb{F}_q$ — First ingredient
5. Compute the coefficients of $(f - f_*) \bmod x^p - 1$ in $\mathbb{Z}/q^{2k}\mathbb{Z}$ — Second ingredient
6. Compute the collision-free exponents of $(f - f_*)$ (+ some noise) — Third ingredient
7. Update f_*
8. Return f_*

Theorem

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

Given a modular black box for $f \in \mathbb{Z}[x]$ and bounds τ, δ, γ, the algorithm returns the sparse representation of f w.h.p. in $O(\tau)$ evaluations and $\tilde{O}(\tau(\log \delta + \log \gamma))$ bit operations

Extensions

Remove sparsity bound

- Given $(\alpha_j)_{j\geq 0}$, find its minimal polynomial without any bound on its degree
  - *Berlekamp–Massey with early termination* [Kaltofen-Lee (2003)]
  - Works over $\mathbb{F}_q$ with $q = \Omega(\delta^4)$ — $2t$ eval. and $\tilde{O}(t \log q)$
- Over $\mathbb{Z}$: early termination modulo $q = \Omega(\delta^4)$
  - Too costly to generate such a prime — $O(\log^3 \delta)$
  - *Random primes without primality testing* [Giorgi-G.-Perret du Cray-Roche (2022)]

The multivariate case

- $f \in \mathbb{Z}[x_0, \ldots, x_{n-1}] \mapsto f_u = f(x, x^{\delta}, x^{\delta^2}, \ldots, x^{\delta^{n-1}}) \in \mathbb{Z}[x]$ — Kronecker (1882)
  - Invertible map
  - $f_u^{\circ} < \delta^n$, $(f_u)_{\#} = f_{\#}$, $(f_u)_{\infty} = f_{\infty}$
  - Evaluation $f_u(\alpha)$: compute $\alpha^{\delta}, \ldots, \alpha^{\delta^{n-1}}$ and $\boxed{f}(\alpha, \ldots, \alpha^{\delta^{n-1}})$

Main result

[Giorgi-G.-Perret du Cray-Roche (2024)]

Given a modular black box $\boxed{f} \in \mathbb{Z}[x_0, \ldots, x_{n-1}]$, $\delta > f^{\circ}$ *and* $\gamma > f_{\infty}$, *one can compute the sparse representation of* f *in* $O(f_{\#})$ *evaluations and* $\tilde{O}(f_{\#}(n \log \delta + \log \gamma))$ *bit operations*

Conclusion

Sparse interpolation over the integers

- Interpolate f from a modular black box in quasi-linear time
- Corollaries for sparse polynomials:
  - Quasi-linear multiplication algorithm [Giorgi-G.-Perret du Cray (2020)]
  - Quasi-linear exact division algorithm [Giorgi-G.-Perret du Cray-Roche (2021-22)]

Open problem: sparse interpolation over finite fields

- Best algorithms: $\tilde{\mathcal{O}}(\tau \log \delta \log q)$ [Huang (2019)]
  - requires $\mathrm{char}(\mathbb{F}_q) > f^\circ$
  - not quasi-linear
- Smaller characteristics: no exponent embedding

Conclusion

Sparse interpolation over the integers

- Interpolate f from a modular black box in quasi-linear time
- Corollaries for sparse polynomials:
  - Quasi-linear multiplication algorithm [Giorgi-G.-Perret du Cray (2020)]
  - Quasi-linear exact division algorithm [Giorgi-G.-Perret du Cray-Roche (2021-22)]

Open problem: sparse interpolation over finite fields

- Best algorithms: $\tilde{\mathcal{O}}(\tau \log \delta \log q)$ [Huang (2019)]
  - requires $\mathrm{char}(\mathbb{F}_q) > f^\circ$
  - not quasi-linear
- Smaller characteristics: no exponent embedding

Thank you!