Fast polynomial computations with space constraints

Calculs polynomiaux rapides avec contraintes de mémoire

Bruno Grenet

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES - INFORMATIQUE

UGA Université Grenoble Alpes

Habilitation à diriger les recherches

13 novembre 2025

Research area: Algebraic computing

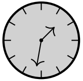
Mathematical computing

- Numerical computing — *approximation of real & complex numbers*
- Algebraic computing — *exactly represented algebraic objects*

Objects in algebraic computing

- Integers, rational numbers, modular rings
- Polynomials, power series, matrices
- Polynomial systems, differential equations, …

$0, 5, -2, \ldots$ $\quad \frac{1}{2}, \frac{2}{5}, -\frac{7}{3}, \ldots$



$3x^2 + 2x - 1$ $\quad \begin{pmatrix} 1 & 6 & 2 \\ -3 & 1 & 4 \\ 12 & 0 & 7 \end{pmatrix}$

Research area: Algebraic computing

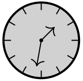
Mathematical computing

- Numerical computing — *approximation of real & complex numbers*
- Algebraic computing — *exactly represented algebraic objects*

Objects in algebraic computing

- Integers, rational numbers, modular rings
- Polynomials, power series, matrices
- Polynomial systems, differential equations, ...

$0, 5, -2, \ldots$ $\quad \frac{1}{2}, \frac{2}{5}, -\frac{7}{3}, \ldots$



$3x^2 + 2x - 1$ $\quad \begin{pmatrix} 1 & 6 & 2 \\ -3 & 1 & 4 \\ 12 & 0 & 7 \end{pmatrix}$

Investigations

- Algorithms & complexity
- Software development
- Mathematics

Some applications

- Security of data and communications
  error correction, cryptography
- Combinatorics, experimental mathematics
- Control theory, robotics
- Modelling (geometry, biology, ...)

Research area: Algebraic computing

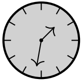
Mathematical computing

- Numerical computing — *approximation of real & complex numbers*
- Algebraic computing — *exactly represented algebraic objects*

Objects in algebraic computing

- Integers, rational numbers, modular rings
- Polynomials, power series, matrices
- Polynomial systems, differential equations, …

$0, 5, -2, \ldots$ $\frac{1}{2}, \frac{2}{5}, -\frac{7}{3}, \ldots$



$3x^2 + 2x - 1$ $\begin{pmatrix} 1 & 6 & 2 \\ -3 & 1 & 4 \\ 12 & 0 & 7 \end{pmatrix}$

Investigations

- Algorithms & complexity
- Software development
- Mathematics

Some applications

- Security of data and communications
  error correction, cryptography
- Combinatorics, experimental mathematics
- Control theory, robotics
- Modelling (geometry, biology, …)

Polynomial computations and space constraints

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) = 2x^6 + 3x^5 + 4x^4 + 2x^3 + 3x + 2 \in \mathbb{Z}/10\mathbb{Z}[x]$$

Polynomial computations and space constraints

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) = 2x^6 + 3x^5 + 4x^4 + 2x^3 + 3x + 2 \in \mathbb{Z}/10\mathbb{Z}[x]$$

Space complexity

- Space required to store intermediate results, in addition to inputs and output
- Large space may hinder the efficiency

Algorithms with small space complexity

Polynomial computations and space constraints

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) = 2x^6 + 3x^5 + 4x^4 + 2x^3 + 3x + 2 \in \mathbb{Z}/10\mathbb{Z}[x]$$

Space complexity

- Space required to store intermediate results, in addition to inputs and output
- Large space may hinder the efficiency

Algorithms with small space complexity

Sparse polynomials

$$5x^6 + 0x^5 + 0x^4 + 2x^3 + 0x^2 - x + 7$$

$$\downarrow$$

$$\{(6,5),(3,2),(1,-1),(0,7)\}$$

Algorithms for sparse polynomials

Polynomial multiplication: 1. The classical algorithm

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) \in \mathbb{Z}/10\mathbb{Z}[x]$$

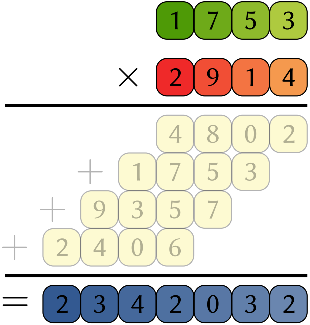
				1	7	5	3
×				2	9	1	4
				4	8	0	2
+			1	7	5	3	
+		9	3	5	7		
+	2	4	0	6			
=	2	3	4	2	0	3	2

Complexity

- 16 products $\rightarrow n^2$ products

Polynomial multiplication: 1. The classical algorithm

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) \in \mathbb{Z}/10\mathbb{Z}[x]$$



Constant space

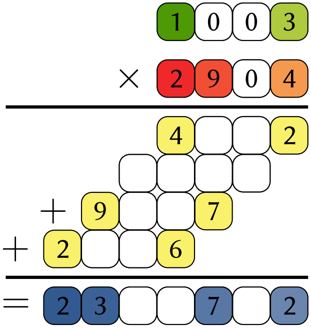
- Compute each product in the output
- Iteratively accumulate the results

Complexity

- 16 products $\rightarrow n^2$ products

Polynomial multiplication: 1. The classical algorithm

$$(x^3 + 3) \times (2x^3 + 9x^2 + 4) \in \mathbb{Z}/10\mathbb{Z}[x]$$



Complexity

- 16 products $\to n^2$ products

Constant space

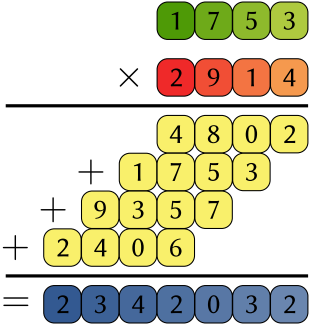
- Compute each product in the output
- Iteratively accumulate the results

Sparse polynomials

- Compute only relevant products
- t nonzero terms $\to t^2$ products

Polynomial multiplication: 1. The classical algorithm

$$(x^3 + 7x^2 + 5x + 3) \times (2x^3 + 9x^2 + x + 4) \in \mathbb{Z}/10\mathbb{Z}[x]$$



Complexity

- 16 products → n^2 products

Constant space

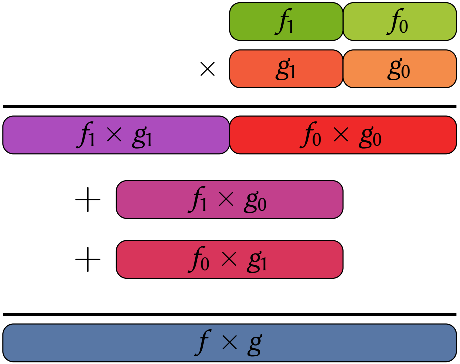
- Compute each product in the output
- Iteratively accumulate the results

Sparse polynomials

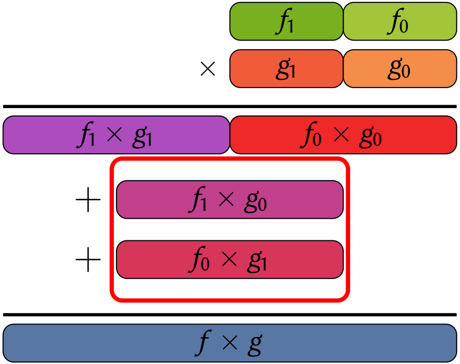
- Compute only relevant products
- t nonzero terms → t^2 products

The classical algorithm handles space constraints easily

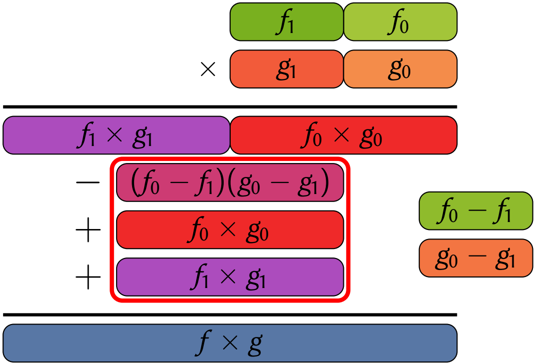
Polynomial multiplication: 2. Karatsuba's algorithm (1962)



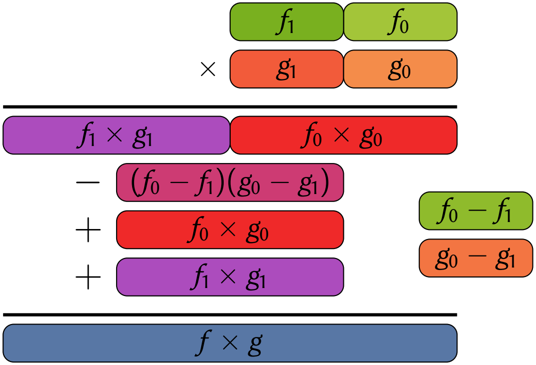
Polynomial multiplication: 2. Karatsuba's algorithm (1962)

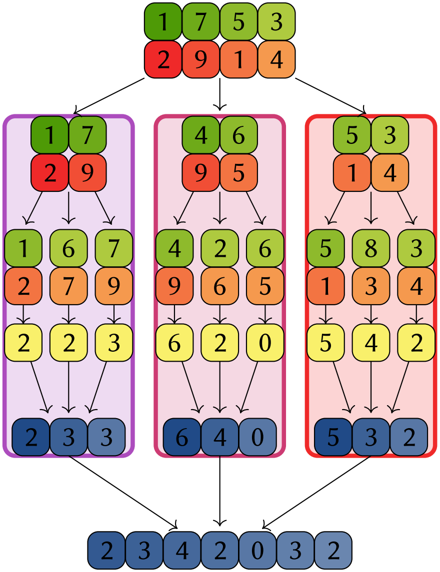


Polynomial multiplication: 2. Karatsuba's algorithm (1962)



Polynomial multiplication: 2. Karatsuba's algorithm (1962)



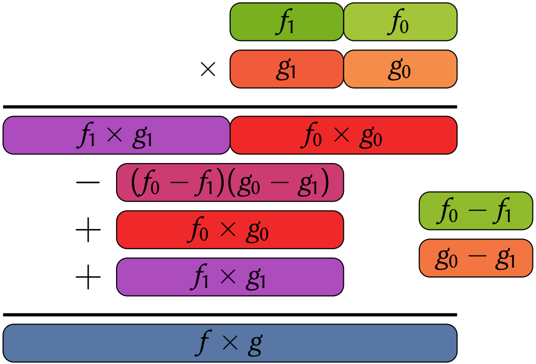


Complexity

- ~~16~~ 9 products

  $\rightarrow n^{\log_2 3} \simeq n^{1.585}$ products

Polynomial multiplication: 2. Karatsuba's algorithm (1962)



Linear space

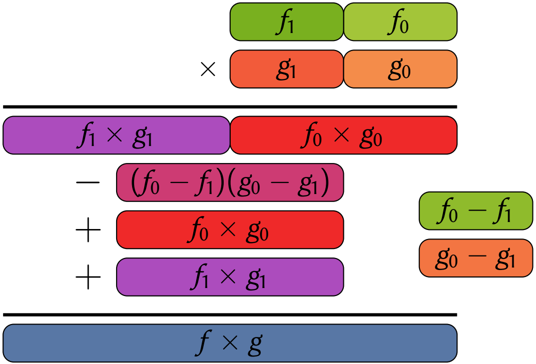
- Store $f_0 - f_1$, $g_0 - g_1$
- Store $f_0 \times g_0$, $f_1 \times g_1$

Complexity

- ~~16~~ 9 products

$\rightarrow n^{\log_2 3} \simeq n^{1.585}$ products

Polynomial multiplication: 2. Karatsuba's algorithm (1962)



Linear space

- Store $f_0 - f_1$, $g_0 - g_1$
- Store $f_0 \times g_0$, $f_1 \times g_1$

Sparse polynomials

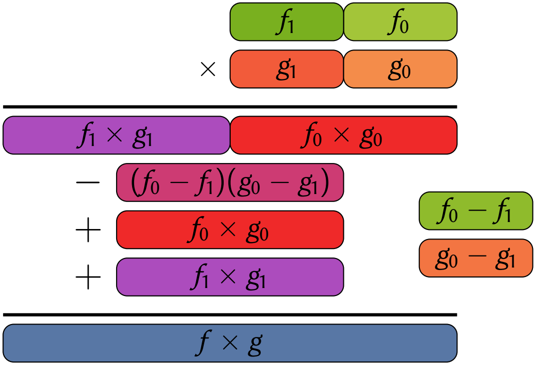
- Possibly *dense* intermediate results
- t nonzero terms $\to \gg t^2$ products

Complexity

- ~~16~~ 9 products

$\to n^{\log_2 3} \simeq n^{1.585}$ products

Polynomial multiplication: 2. Karatsuba's algorithm (1962)



Linear space

- Store $f_0 - f_1$, $g_0 - g_1$
- Store $f_0 \times g_0$, $f_1 \times g_1$

Sparse polynomials

- Possibly *dense* intermediate results
- t nonzero terms $\to \gg t^2$ products

Karatsuba's algorithm *does not* handle space constraints easily

Complexity

- ~~16~~ 9 products

$$\to n^{\log_2 3} \simeq n^{1.585} \text{ products}$$

Can we combine *fast algorithms* with *space constraints* in polynomial computations?

Part I. Time- and space-efficient computations

- Fast algorithms with (close to) constant space
  - Polynomial: multiplication, Euclidean division, evaluation, interpolation
  - Power series: multiplication, inversion, division
  - Matrix: multiplication, system solving, …
- What is space complexity of functions?

Part II. Sparse polynomial computations

- Quasi-linear time sparse interpolation
  - Multiplication and exact division
  - Verification
- Polynomial-time low-degree factorization and (partial) divisibility test

Can we combine *fast algorithms* with
space constraints in polynomial computations?

Part I. Time- and space-efficient computations

- Fast algorithms with (close to) constant space
  - Polynomial: multiplication, Euclidean division, evaluation, interpolation
  - Power series: multiplication, inversion, division
  - Matrix: multiplication, system solving, …
- What is space complexity of functions?

Part II. Sparse polynomial computations

- Quasi-linear time sparse interpolation
  - Multiplication and exact division
  - Verification
- Polynomial-time low-degree factorization and (partial) divisibility test

I. Time- and space-efficient computations

What is space complexity?

Algebraic RAM

- Algebraic registers containing one ring element each — for some ring R
- Registers for pointers of size $O(\log n)$ — $n = \#$ input registers

Space complexity

- Number of registers used, not counting the input and output registers
- Distinction between algebraic registers and pointers

Relation with standard complexity classes

- Depends on the *relative* size of n and $\#\mathrm{R}$
- Assuming $\log(n) \simeq \log(\#\mathrm{R})$:
  - constant number of registers of both kinds $\simeq$ complexity class FL
  - $O(1)$ algebraic registers, $O(\log n)$ pointers $\simeq$ complexity class $\mathrm{FSPACE}(\log^2 n)$

Permission models

ro/wo: read-only inputs & write-only output

- 👍 classical model in complexity theory
- 👎 further from practice, multiplication requires $\Omega(n^2)$ time × space [Abrahamson (1985)]

ro/rw: read-only inputs & read-write output

- 👍 closer to practice, allows parallel access to the inputs
- 👎 somewhat restrictive in a sequential model

rw/rw: read-write inputs & read-write output, *inputs restored at the end*

- 👍 still consistent with practice, allows recursive calls / use as subroutines
- 👎 not suitable for parallel programming

Goal: time- and space-efficient algorithms in the ro/rw and rw/rw models

- Multiplication
- Euclidean division

Results for multiplication algorithms

algorithm	model	time	alg. sp.	# pointers	
Classical	ro/wo	$O(n^2)$	$O(1)$	$O(1)$	folklore
Karatsuba	ro/wo	$O(n^{\log 3})$	$O(n)$	$O(\log n)$	Karatsuba (1962)
	ro/rw		n	$O(\log n)$	Thomé (2002)
	ro/rw		$O(1)$	$O(\log n)$	Roche (2009)
Toom-Cook	ro/wo	$O(n^{\log_3 5})$	$O(n)$	$O(\log n)$	Toom (1963)
FFT-based (given $\omega^{2n} = 1$)	ro/wo	$O(n \log n)$	$O(n)$	$O(1)$	Cooley, Tukey (1965)
	ro/rw		$O(1)$	$O(1)$	Roche (2009) (if $n = 2^k$)
	ro/rw		$O(1)$	$O(1)$	Harvey, Roche (2010)

Results for multiplication algorithms

algorithm	model	time	alg. sp.	# pointers	
Classical	ro/wo	$O(n^2)$	$O(1)$	$O(1)$	folklore
Karatsuba	ro/wo	$O(n^{\log 3})$	$O(n)$	$O(\log n)$	Karatsuba (1962)
	ro/rw		n	$O(\log n)$	Thomé (2002)
	ro/rw		$O(1)$	$O(\log n)$	Roche (2009)
Toom-Cook	ro/wo	$O(n^{\log_3 5})$	$O(n)$	$O(\log n)$	Toom (1963)
FFT-based (given $\omega^{2n} = 1$)	ro/wo	$O(n \log n)$	$O(n)$	$O(1)$	Cooley, Tukey (1965)
	ro/rw		$O(1)$	$O(1)$	Roche (2009) (if $n = 2^k$)
	ro/rw		$O(1)$	$O(1)$	Harvey, Roche (2010)

Our result

[Giorgi, G., Roche (2019)]

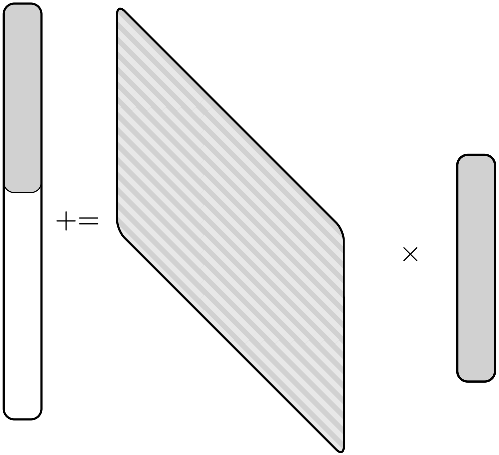
Any linear-space multiplication algorithm can be made constant-space with the same asymptotic time complexity in the ro/rw model

$h = f \times g$ as a matrix-vector product

$$
\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \\ h_{10} \end{bmatrix}
=
\begin{bmatrix}
f_0 & & & & & \\
f_1 & f_0 & & & 0 & \\
f_2 & f_1 & f_0 & & & \\
f_3 & f_2 & f_1 & f_0 & & \\
f_4 & f_3 & f_2 & f_1 & f_0 & \\
f_5 & f_4 & f_3 & f_2 & f_1 & f_0 \\
 & f_5 & f_4 & f_3 & f_2 & f_1 \\
 & & f_5 & f_4 & f_3 & f_2 \\
 & & & f_5 & f_4 & f_3 \\
 & 0 & & & f_5 & f_4 \\
 & & & & & f_5
\end{bmatrix}
\times
\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{bmatrix}
$$

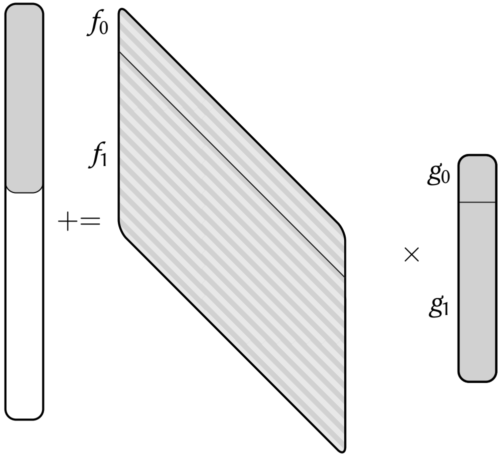
$$h = f \times g$$

Proof sketch: *semi-cumulative* algorithm



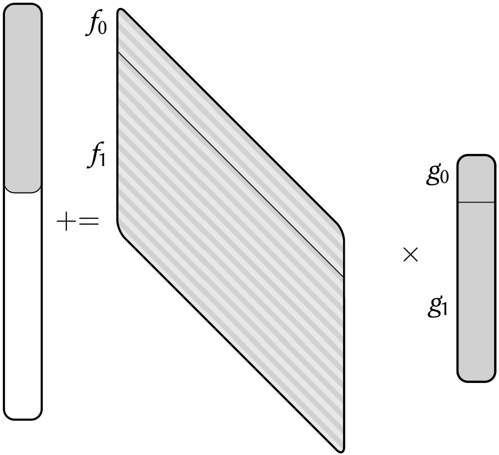
$$h \mathrel{+}= f \times g$$

Proof sketch: *semi-cumulative* algorithm



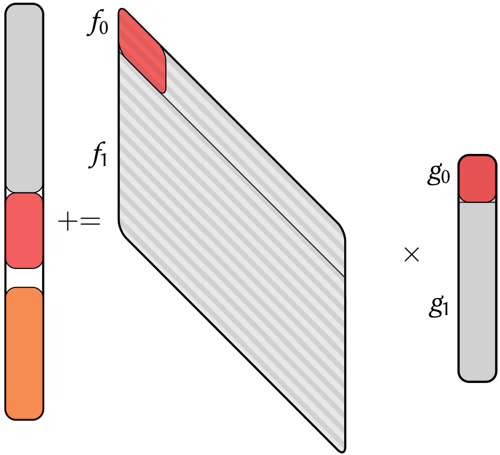
$$h \mathrel{+}= (f_0 + X^k f_1) \times (g_0 + X^k g_1)$$

Proof sketch: *semi-cumulative* algorithm



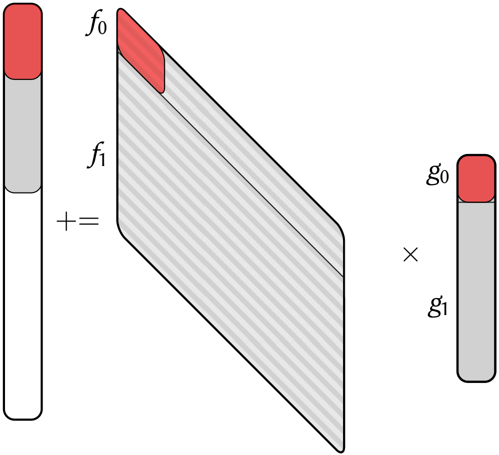
$$h \mathrel{+}= f_0 g_0 + X^k (f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Proof sketch: *semi-cumulative* algorithm



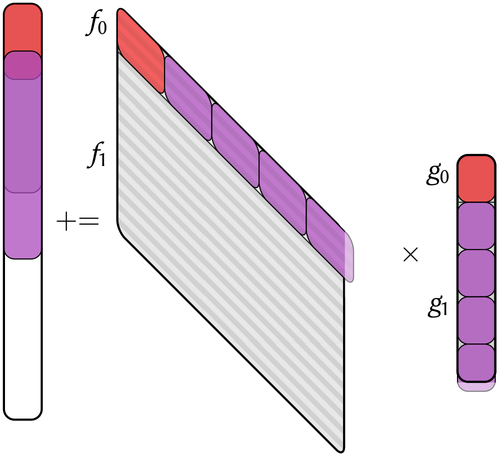
$$h \mathrel{+}= f_0 g_0 + X^k(f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Proof sketch: *semi-cumulative* algorithm



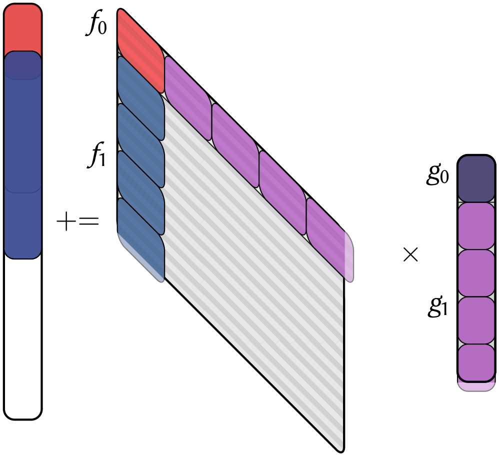
$$h \mathrel{+}= f_0 g_0 + X^k (f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Proof sketch: *semi-cumulative* algorithm



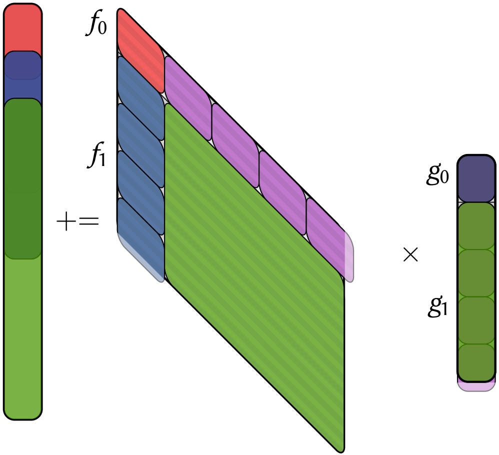
$$h += f_0 g_0 + X^k (f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Proof sketch: *semi-cumulative* algorithm



$$h \mathrel{+}= f_0 g_0 + X^k (f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Proof sketch: *semi-cumulative* algorithm



$$h \mathrel{+}= f_0 g_0 + X^k (f_0 g_1 + f_1 g_0) + X^{2k} f_1 g_1$$

Cumulative and in-place operations – $\texttt{rw}/\texttt{rw}$ model

Three kinds of operations

Standard: $h := f \times g$: $(f, g, h) \mapsto (f, g, f \times g)$

Cumulative: $h += f \times g$: $(f, g, h) \mapsto (f, g, h + f \times g)$

In-place: $f *= g$: $(f, g) \mapsto (f \times g, g)$

Our results

[Dumas, G. (2024 & 2026)]

problem	model	time	alg. sp.	# pointers
$h += f \times g$ *(general)*	rw/rw	$O(\mathsf{M}(n))$	$O(1)$	$O(\log n)$
$h += f \times g$ *(with FFT)*	rw/rw	$O(\mathsf{M}(n))$	$O(1)$	$O(1)$
$h += f \times g \bmod x^n$	rw/rw	$O(\mathsf{M}(n))$	$O(1)$	$O(1)$
$f *= g \bmod x^n$	rw/rw	$O(\mathsf{M}(n) \log n)$	$O(1)$	$O(\log n)$
$f /= g \bmod x^n$	rw/rw	$O(\mathsf{M}(n) \log n)$	$O(1)$	$O(\log n)$

Results for Euclidean division

algorithm	model	output	time	alg. sp.	
Classical algorithm	`ro/wo`	Quotient + Remainder	$O(n^2)$	$O(1)$	
		Quotient only			Monagan (1993)
		Remainder only			Monagan (1993)
Fast algorithm	`ro/wo`	Quotient + Remainder	$O(\mathsf{M}(n))$	$O(n)$	Strassen (1973), …
		Quotient only			
		Remainder only			

Results for Euclidean division

algorithm	model	output	time	alg. sp.	
Classical algorithm	ro/wo	Quotient + Remainder	$O(n^2)$	$O(1)$	Monagan (1993)
		Quotient only			
		Remainder only			Monagan (1993)
Fast algorithm	ro/wo	Quotient + Remainder	$O(\mathsf{M}(n))$	$O(n)$	Strassen (1973), …
		Quotient only			
		Remainder only			

Our results

algorithm	model	output	time	alg. sp.	
Fast and low space	ro/rw	Quotient + Remainder	$O(\mathsf{M}(n))$	$O(1)$	Giorgi, G., Roche (2020)
		Quotient only	$O(\mathsf{M}(n) \log n)$	$O(1)$	
		Remainder only	$O(\mathsf{M}(n))$	$n + O(1)$	
	rw/rw	Remainder only	$O(\mathsf{M}(n) \log n)$	$O(1)$	Dumas G. (2024)

Fast Euclidean division

[Strassen (1973), …]

$$
\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \end{bmatrix}
=
\begin{bmatrix}
g_0 & & & & & \\
g_1 & g_0 & & & 0 & \\
g_2 & g_1 & g_0 & & & \\
g_3 & g_2 & g_1 & g_0 & & \\
g_4 & g_3 & g_2 & g_1 & g_0 & \\
g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\
 & g_5 & g_4 & g_3 & g_2 & g_1 \\
 & & g_5 & g_4 & g_3 & g_2 \\
 & & & g_5 & g_4 & g_3 \\
 & 0 & & & g_5 & g_4 \\
 & & & & & g_5
\end{bmatrix}
\times
\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}
+
\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

$$f = g \times q + r$$

Fast Euclidean division

[Strassen (1973), …]

$$
\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \end{bmatrix}
=
\begin{bmatrix}
g_0 & & & & & \\
g_1 & g_0 & & 0 & & \\
g_2 & g_1 & g_0 & & & \\
g_3 & g_2 & g_1 & g_0 & & \\
g_4 & g_3 & g_2 & g_1 & g_0 & \\
g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\
 & g_5 & g_4 & g_3 & g_2 & g_1 \\
 & & g_5 & g_4 & g_3 & g_2 \\
 & & & g_5 & g_4 & g_3 \\
 & 0 & & & g_5 & g_4 \\
 & & & & & g_5
\end{bmatrix}
\times
\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}
+
\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

$$f = g \times q + r$$

Fast Euclidean division

[Strassen (1973), …]

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} g_0 & & & & & \\ g_1 & g_0 & & 0 & & \\ g_2 & g_1 & g_0 & & & \\ g_3 & g_2 & g_1 & g_0 & & \\ g_4 & g_3 & g_2 & g_1 & g_0 & \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_4 \\ q_5 \end{bmatrix} + \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$$

$$\begin{bmatrix} f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \end{bmatrix} = \begin{bmatrix} g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\ & g_5 & g_4 & g_3 & g_2 & g_1 \\ & & g_5 & g_4 & g_3 & g_2 \\ & & & g_5 & g_4 & g_3 \\ & 0 & & & g_5 & g_4 \\ & & & & & g_5 \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f = g \times q + r$$

Fast Euclidean division

[Strassen (1973), …]

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} g_0 & & & & \\ g_1 & g_0 & & 0 & \\ g_2 & g_1 & g_0 & & \\ g_3 & g_2 & g_1 & g_0 & \\ g_4 & g_3 & g_2 & g_1 & g_0 \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} + \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$$

$$\begin{bmatrix} f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \end{bmatrix} = \begin{bmatrix} g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\ & g_5 & g_4 & g_3 & g_2 & g_1 \\ & & g_5 & g_4 & g_3 & g_2 \\ & & & g_5 & g_4 & g_3 \\ & 0 & & & g_5 & g_4 \\ & & & & & g_5 \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}$$

$$f = g \times q + r$$

Fast Euclidean division

[Strassen (1973), . . .]

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} - \begin{bmatrix} g_0 & & & & \\ g_1 & g_0 & & 0 & \\ g_2 & g_1 & g_0 & & \\ g_3 & g_2 & g_1 & g_0 & \\ g_4 & g_3 & g_2 & g_1 & g_0 \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix} = \begin{bmatrix} g_5 & g_4 & g_3 & g_2 & g_1 & g_0 \\ & g_5 & g_4 & g_3 & g_2 & g_1 \\ & & g_5 & g_4 & g_3 & g_2 \\ & & & g_5 & g_4 & g_3 \\ & 0 & & & g_5 & g_4 \\ & & & & & g_5 \end{bmatrix}^{-1} \times \begin{bmatrix} f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \end{bmatrix}$$

$$f = g \times q + r$$

Fast Euclidean division

[Strassen (1973), …]

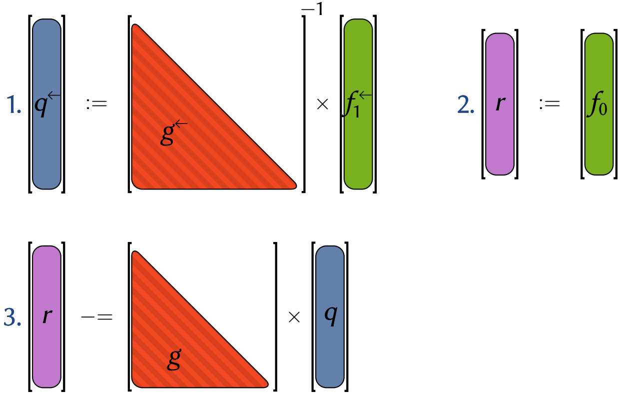
$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} - \begin{bmatrix} g_0 & & & & \\ g_1 & g_0 & & 0 & \\ g_2 & g_1 & g_0 & & \\ g_3 & g_2 & g_1 & g_0 & \\ g_4 & g_3 & g_2 & g_1 & g_0 \end{bmatrix} \times \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

$$\begin{bmatrix} q_5 \\ q_4 \\ q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} = \begin{bmatrix} g_5 & & & & & \\ g_4 & g_5 & & 0 & & \\ g_3 & g_4 & g_5 & & & \\ g_2 & g_3 & g_4 & g_5 & & \\ g_1 & g_2 & g_3 & g_4 & g_5 & \\ g_0 & g_1 & g_2 & g_3 & g_4 & g_5 \end{bmatrix}^{-1} \times \begin{bmatrix} f_{10} \\ f_9 \\ f_8 \\ f_7 \\ f_6 \\ f_5 \end{bmatrix}$$

$$q^{\leftarrow} = f^{\leftarrow} / g^{\leftarrow} \bmod x^n$$

$$r = (f - g \times q \bmod x^{n-1})$$

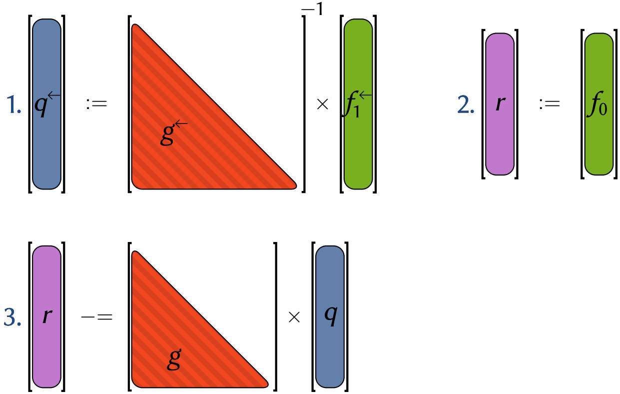
Euclidean division algorithms



problem	model	time	alg. sp.	# pointers
$q := f \operatorname{div} g,\ r := f \bmod g$	`ro/rw`	$O(\mathsf{M}(n))$	$O(1)$	$O(1)$

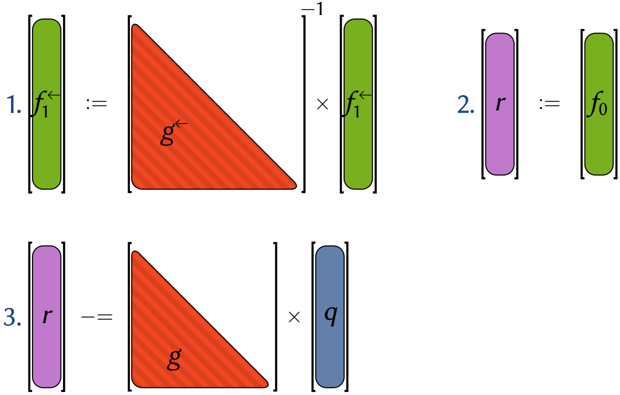
Giorgi G. Roche (2020)

Euclidean division algorithms



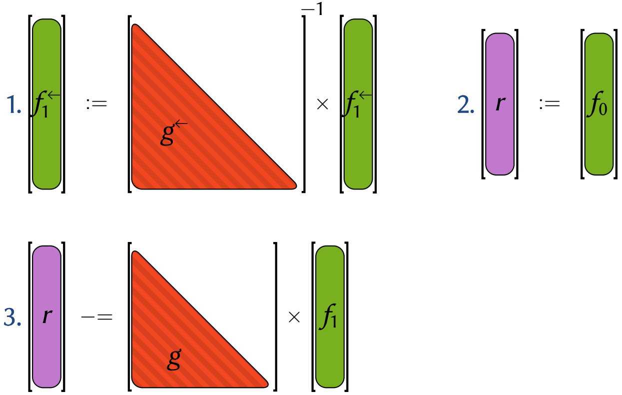
problem	model	time	alg. sp.	# pointers	
$r := f \bmod g$	ro/rw	$O(\mathsf{M}(n))$	$n + O(1)$	$O(1)$	Giorgi G. Roche (2020)

Euclidean division algorithms



problem	model	time	alg. sp.	# pointers	
$r := f \bmod g$	ro/rw	$O(\mathsf{M}(n))$	$n + O(1)$	$O(1)$	Giorgi G. Roche (2020)

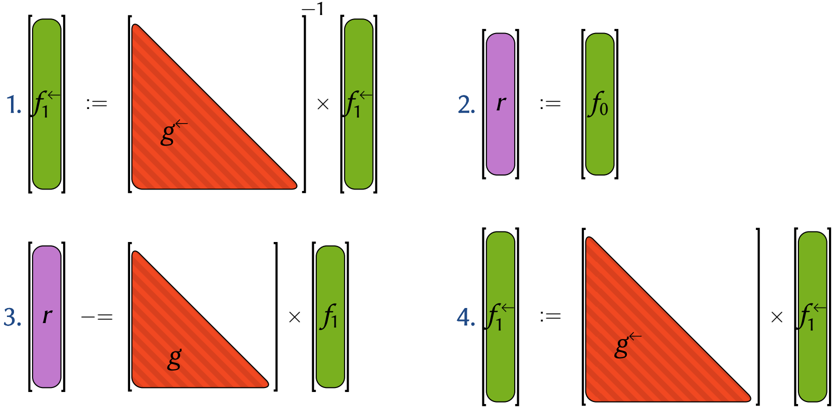
Euclidean division algorithms



problem	model	time	alg. sp.	# pointers
$r := f \bmod g$	ro/rw	$O(\mathsf{M}(n))$	$n + O(1)$	$O(1)$

Giorgi G. Roche (2020)

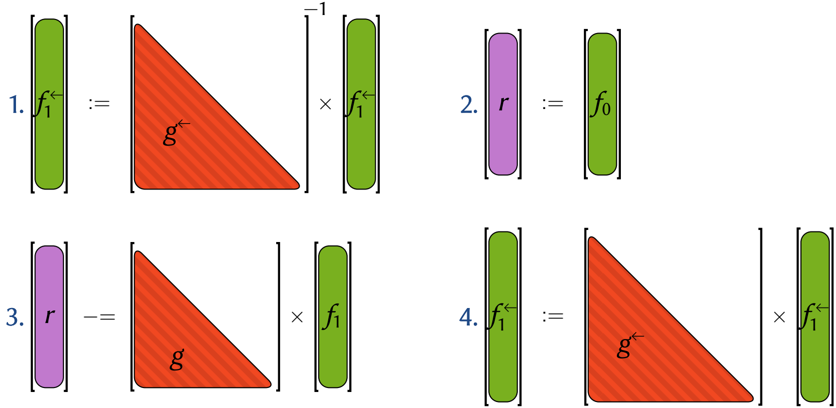
Euclidean division algorithms



problem	model	time	alg. sp.	# pointers
$r := f \bmod g$	ro/rw	$O(\mathsf{M}(n))$	$n + O(1)$	$O(1)$

Giorgi G. Roche (2020)

Euclidean division algorithms



problem	model	time	alg. sp.	# pointers
$r := f \bmod g$	rw/rw	$O(\mathsf{M}(n)\log n)$	$O(1)$	$O(\log n)$

Dumas G. (2024 & 2026)

Summary

Time- and space-efficient polynomial computations

- Many fast algorithms can be made
  - (almost) constant-space
  - with (almost) the same asymptotic time complexity
- Requires to move away from the classical `ro/wo` model
- Promising results in practice

Automatic derivations

[Dumas G. (2024 & 2026)]

- Make bilinear algorithms constant-space automatically in the `rw/rw` model
- Application: constant-space Strassen's alg., fast in-place linear algebra

Open problems

- Improved complexities, `rw/rw` $\to$ `ro/rw`, other operations such as GCD, …
- Right models for time-space complexity of functions?
- Can you replace f g by $f \times g$ without extra space? [Roche'09]

II. Sparse polynomial computations

Sparse representation of polynomials

$$\to \left\{(1,1),(9,4),(10,6),(16,7),(20,8),(25,3),(30,1)\right\}$$

Sparse representation of polynomials

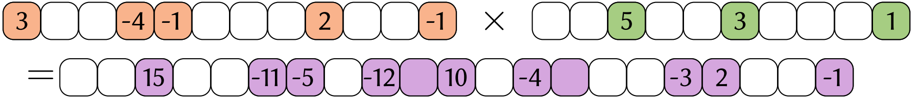
$$\to \left\{(1,1),(9,4),(10,6),(16,7),(20,8),(25,3),(30,1)\right\}$$

$$f = \sum_{i=0}^{t-1} c_i x^{e_i} \in \mathsf{R}[x] \qquad \to \qquad \left\{(c_i, e_i) : 0 \le i < t, c_i \neq 0\right\}$$

Notations

R:	ring of coefficients	
f°:	degree of f	$\max_i e_i$
$f_\#$:	*sparsity* of f	t
f_∞:	*height* of f if $\mathsf{R} = \mathbb{Z}$	$\max \lvert c_i \rvert$
	q if $\mathsf{R} = \mathbb{F}_q$	
S_f:	*support* of f	$\{e_i : 0 \le i < t\}$
size(f):	$f_\#(\log f^\circ + \log f_\infty)$	

Sparse polynomial multiplication

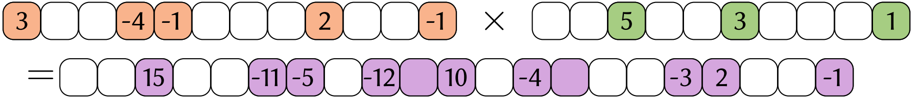


Problem

- Compute $h = f \times g$, with
  - $t = \max(f_\#, g_\#)$, $d = \max(f^\circ, g^\circ)$, $m = \max(f_\infty, g_\infty)$
  - $s = \#(S_f + S_g)$ *structural sparsity*: $h_\# \leq s \leq f_\# g_\#$

algorithm	ring	time	
Classical	any	$O(t^2 \log t)$	folklore
Quadratic	any	$O(t^2)$	Johnson (1974)
Output-sensitive	any	$\tilde{O}(s \log m + t \log d)$	Arnold Roche (2015)

Sparse polynomial multiplication



Problem

- Compute $h = f \times g$, with
  - $t = \max(f_\#, g_\#)$, $d = \max(f^\circ, g^\circ)$, $m = \max(f_\infty, g_\infty)$
  - $s = \#(S_f + S_g)$ *structural sparsity*: $h_\# \leq s \leq f_\# g_\#$

algorithm	ring	time	
Classical	any	$O(t^2 \log t)$	folklore
Quadratic	any	$O(t^2)$	Johnson (1974)
Output-sensitive	any	$\tilde{O}(s \log m + t \log d)$	Arnold Roche (2015)

Our result

[Giorgi, G., Perret du Cray (2020)]

Quasi-linear	$\mathbb{Z}$	$\tilde{O}(\tau \log m + \tau \log d)$	where $\tau = \max(f_\#, g_\#, h_\#)$

The main tool: Sparse interpolation

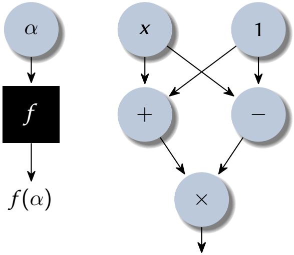
General definition

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 $f = \sum_{i=0}^{t-1} c_i x^{e_i}$

Many variants

- Ring of coefficients: $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \ \mathbb{F}_q, \ \mathbb{Z}/n\mathbb{Z}$
- Number of variables: univariate or multivariate
- Input representation:
  - black box
  - straight-line program (SLP) / arithmetic circuit



The main tool: Sparse interpolation

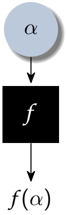
General definition

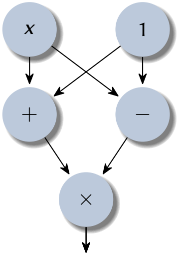
Inputs: A way to *evaluate* a sparse polynomial $f \in R[x]$
Bounds $\delta \geq f^\circ$, $\tau \geq f_\#$, and $\gamma \geq f_\infty$ (optional)

Output: The sparse representation of $f = \sum_{i=0}^{t-1} c_i x^{e_i}$

Many variants

- Ring of coefficients: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{F}_q$, $\mathbb{Z}/n\mathbb{Z}$
- Number of variables: univariate or multivariate
- Input representation:
  - black box
  - straight-line program (SLP) / arithmetic circuit





Black-box sparse interpolation

$$f = \sum_{i=0}^{t-1} c_i x^{e_i} \to f(\omega^j) = \sum_{i=0}^{t-1} c_i (\omega^{e_i})^j$$

Lemma [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 sketch

1. Compute $(f(\omega^j))_{0\leq j<2\tau}$ using f
2. Compute its minimal polynomial Λ, and its roots
3. Get the exponents e_i from the roots ω^{e_i} of Λ
4. Get the coefficients c_i by solving a transposed Vandermonde system

Black-box sparse interpolation

$$f = \sum_{i=0}^{t-1} c_i x^{e_i} \to f(\omega^j) = \sum_{i=0}^{t-1} c_i (\omega^{e_i})^j$$

Lemma [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 sketch

1. Compute $(f(\omega^j))_{0\leq j<2\tau}$ using f
2. Compute its minimal polynomial Λ, and its roots
3. Get the exponents e_i from the roots ω^{e_i} of Λ
4. Get the coefficients c_i by solving a transposed Vandermonde system

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

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

SLP sparse interpolation

From an SLP,

- f can be computed explicitly in time $\tilde{\mathcal{O}}(f^\circ)$ *expression swell*
- $f \bmod x^p - 1 = \sum_i c_i x^{e_i \bmod p}$ can be computed in time $\tilde{\mathcal{O}}(p)$ [Garg-Schost (2009)]

Loss of information

- Exponents only known *modulo p*
  - Embed them into the coefficients using an SLP for f' [Huang (2019)]
- Possible *collisions* between monomials
  - Correct errors using several random primes p [Arnold Giesbrecht Roche (2013)]

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

Given a size-ℓ SLP for $f \in \mathbb{F}_q[x]$, $f^\circ \geq$ char$(\mathbb{F}_q)$, and bounds $\tau \geq f_\#$ and $\delta \geq f^\circ$, one can compute the sparse representation of f in $\tilde{\mathcal{O}}(\ell\tau \log(\delta) \times \log(q))$ bit operations

Our sparse interpolation algorithm

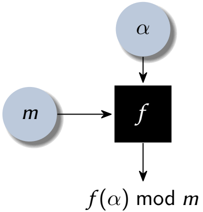
Theorem

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

Given a modular black box for $f \in \mathbb{Z}[x]$ and bounds $\tau \geq f_{\#}$, $\delta \geq f^{\circ}$, $\gamma \geq f_{\infty}$, one can compute the sparse representation of f in $\tilde{\mathcal{O}}(\tau(\log(\delta) + \log(q)))$ bit operations

Modular black box

- Can be implemented given an SLP
- Pure black box: evaluations on $\mathbb{Z} \setminus \{0, \pm 1\}$ have size $\Omega(\delta)$



Our sparse interpolation algorithm

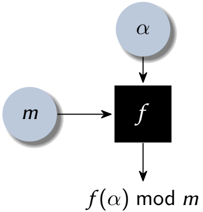
Theorem

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

Given a modular black box for $f \in \mathbb{Z}[x]$ *and bounds* $\tau \geq f_\#, \delta \geq f^\circ, \gamma \geq f_\infty$, *one can compute the sparse representation of* f *in* $\tilde{\mathcal{O}}(\tau(\log(\delta) + \log(q)))$ *bit operations*

Modular black box

- Can be implemented given an SLP
- Pure black box: evaluations on $\mathbb{Z} \setminus \{0, \pm 1\}$ have size $\Omega(\delta)$



Remark

- If $\omega^p = 1$, $f(\omega^j) = f_{[p]}(\omega^j)$ where $f_{[p]} = f \bmod x^p - 1$

Sketch of the algorithm

1. Exponents of $f_{[p]}$: interpolation of f with ω of order p in a small $\mathbb{F}_q$
2. Coefficients of $f_{[p]}$ and $f'_{[p]}$: interpolation of f *and* f' with ω_k of order p in $\mathbb{Z}/q^k\mathbb{Z}$
3. Deduce *some* terms $c_i x^{e_i}$ of f
4. Repeat with several random primes p and q — *deal with collisions*

Back to polynomial multiplication

The problem

Inputs: $f, g \in \mathbb{Z}[x]$ in sparse representation
Output: $h = f \times g$

Approach and difficulty

- Use sparse interpolation to compute h
- Implement the modular black-box using f and g
- The algorithms require a bound $\tau \geq h_{\#}$

The solution

- Use an *a priori* bound
- Check the result *a posteriori*, and increase the bound if it is incorrect

New problem: product verification

Inputs: $f, g, h \in \mathbb{Z}[x]$ in sparse representation
Output: Does $h = f \times g$?

Verification of polynomial product $h = f \times g$

Classical approach

1. Sample a random element $\alpha \in \mathrm{R}$
2. Return $h(\alpha) \stackrel{?}{=} f(\alpha) \times g(\alpha)$
   - Works if R is large enough
   - If $\mathrm{R} = \mathbb{Z}$, check the result *modulo* a random prime q

Case of sparse polynomials

- Too costly approach:
  - Evaluation of f on $\alpha \in \mathbb{Z}_{\neq 0, \pm 1}$ has cost $\Omega(f^\circ)$ *output size*
  - Evaluation of f on $\alpha \in \mathbb{Z}/q\mathbb{Z}$ has cost $\tilde{\mathcal{O}}(t \log(f^\circ) \log(q))$ *binary exponentation*
- Solution: check $h = f \times g \bmod (x^p - 1)$ for some random prime p

Modular product evaluation

Input: $f, g \in \mathbb{Z}[x]$, $p, q \in \mathbb{Z}_{>0}$, $\alpha \in \mathbb{Z}/q\mathbb{Z}$
Output: $(f \times g \bmod x^p - 1)(\alpha)$ without computing $f \times g \bmod x^p - 1$

Modular product evaluation

$$f \times g$$

$$\begin{bmatrix} f_0 & & & & \\ f_1 & f_0 & & & \\ f_2 & f_1 & f_0 & & \\ f_3 & f_2 & f_1 & f_0 & \\ f_4 & f_3 & f_2 & f_1 & f_0 \\ & f_4 & f_3 & f_2 & f_1 \\ & & f_4 & f_3 & f_2 \\ & & & f_4 & f_3 \\ & & & & f_4 \end{bmatrix} \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

Modular product evaluation

$$f \times g \bmod (x^p - 1)$$

$$\begin{bmatrix} f_0 & f_4 & f_3 & f_2 & f_1 \\ f_1 & f_0 & f_4 & f_3 & f_2 \\ f_2 & f_1 & f_0 & f_4 & f_3 \\ f_3 & f_2 & f_1 & f_0 & f_4 \\ f_4 & f_3 & f_2 & f_1 & f_0 \end{bmatrix} \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

Modular product evaluation

$$\Big(f \times g \bmod (x^p - 1)\Big)(\alpha)$$

$$\begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 \end{bmatrix} \times \left[\begin{bmatrix} f_0 & f_4 & f_3 & f_2 & f_1 \\ f_1 & f_0 & f_4 & f_3 & f_2 \\ f_2 & f_1 & f_0 & f_4 & f_3 \\ f_3 & f_2 & f_1 & f_0 & f_4 \\ f_4 & f_3 & f_2 & f_1 & f_0 \end{bmatrix} \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} \right]$$

Modular product evaluation

$$\Big(f \times g \bmod (x^p - 1)\Big)(\alpha)$$

$$\left[\begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 \end{bmatrix} \times \begin{bmatrix} f_0 & f_4 & f_3 & f_2 & f_1 \\ f_1 & f_0 & f_4 & f_3 & f_2 \\ f_2 & f_1 & f_0 & f_4 & f_3 \\ f_3 & f_2 & f_1 & f_0 & f_4 \\ f_4 & f_3 & f_2 & f_1 & f_0 \end{bmatrix}\right] \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

Modular product evaluation

$$\Big(f \times g \bmod (x^p - 1)\Big)(\alpha)$$

$$\left[\begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 \end{bmatrix} \times \begin{bmatrix} f_0 & f_4 & f_3 & f_2 & f_1 \\ f_1 & f_0 & f_4 & f_3 & f_2 \\ f_2 & f_1 & f_0 & f_4 & f_3 \\ f_3 & f_2 & f_1 & f_0 & f_4 \\ f_4 & f_3 & f_2 & f_1 & f_0 \end{bmatrix} \right] \times \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

Theorem

Modular product evaluation in

- $O(p)$ ring operations [Giorgi (2018)]
- $O(t \log(t \log d))$ ring operations [Giorgi G. Perret du Cray (2023)]

Summary

Fast sparse polynomial computations

- Quasi-linear sparse interpolation *over* $\mathbb{Z}$
- Quasi-linear (sparse) modular product verification *over any ring*
- Quasi-linear sparse multiplication *over* $\mathbb{Z}$

Other consequences

- Quasi-linear sparse *exact division* *over* $\mathbb{Z}$
- Polynomials with *unbalanced coefficients* *over* $\mathbb{Z}$
  - Fast sparse interpolation and multiplication
  - Quasi-linear *dense* multiplication

Open problems

- Extend to finite fields, quasi-linear sparse unbalanced multiplication, ...
- Given $2t$ evaluations of a t-sparse $f \in \mathbb{R}[x]$ on positive reals, how to reconstruct f?
- Given t-sparse $f, g \in \mathbb{R}[x]$, can $fg + 1$ have $\Omega(t^2)$ real roots?

Conclusion & perspectives

Revisit results on polynomial computations

- Behavior in the presence of space constraints
- Questions related to complexity theory
  - Definition(s) of space complexity
  - Frontier P / NP-hard *divisibility/factorization of sparse polynomials*

Constant-space algorithm in quantum computing

- *Reversible* algorithms in the $\mathtt{rw}/\mathtt{rw}$ model *suitable for quantum computing*
- Example: Karatsuba's algorithm with $O(1)$ ancilla qubits [Ottow's internship (2021)]
- Application to Shor's or Regev's factorization algorithms

Sparse interpolation, linear codes, cryptography [Giorgi G. Simkin (2025)]

- Sparse interpolation $\approx$ syndrome decoding of Reed-Solomon codes
- Can be used in *oblivious ciphertext (de)compression* $\rightarrow$ PIR, searchable encryption, …

Conclusion & perspectives

Revisit results on polynomial computations

- Behavior in the presence of space constraints
- Questions related to complexity theory
  - Definition(s) of space complexity
  - Frontier P / NP-hard *divisibility/factorization of sparse polynomials*

Constant-space algorithm in quantum computing

- *Reversible* algorithms in the `rw`/`rw` model *suitable for quantum computing*
- Example: Karatsuba's algorithm with $O(1)$ ancilla qubits [Ottow's internship (2021)]
- Application to Shor's or Regev's factorization algorithms

Sparse interpolation, linear codes, cryptography [Giorgi G. Simkin (2025)]

- Sparse interpolation $\approx$ syndrome decoding of Reed-Solomon codes
- Can be used in *oblivious ciphertext (de)compression* $\rightarrow$ PIR, searchable encryption, …

Thank you!