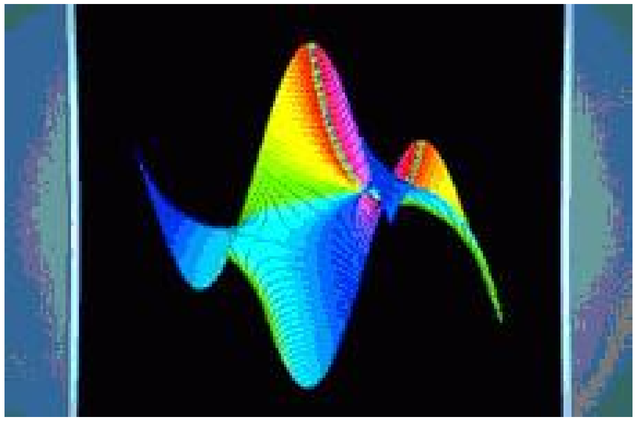
Fast algorithms for polynomials and matrices

Alin Bostan



Algorithms Project, INRIA

JNCF 2010

This course

- Main objects
  - polynomials $\mathbb{K}[x]$
  - rational fractions $\mathbb{K}(x)$
  - power series $\mathbb{K}[[x]]$
  - matrices $\mathcal{M}_r(\mathbb{K})$
  - polynomial matrices $\mathcal{M}_r(\mathbb{K}[x])$
  - power series matrices $\mathcal{M}_r(\mathbb{K}[[x]])$

  where $\mathbb{K}$ is a field (generally supposed of characteristic 0 or large)

- Secondary/auxiliary objects
  - linear recurrences with constant, or polynomial, coefficients
  - linear differential equations with polynomial, or power series, coefficients

This course

- Aims
  - design and analysis of fast algorithms for various algebraic problems
  - Fast = using asymptotically few operations $(+, \times, \div)$ in the basefield $\mathbb{K}$
  - Holy Grail: quasi-optimal algorithms = (time) complexity almost linear in the input/output size

- Specific algorithms depending on the kind of the input
  - dense (i.e., arbitrary)
  - structured (i.e., special relations between coefficients)
  - sparse (i.e., few elements)

- In this lecture, we focus on dense objects

A word about structure and sparsity

- sparse means
  - for degree n polynomials: $s \ll n$ coefficients
  - for $r \times r$ matrices: $s \ll r^2$ entries

- structured means
  - for $r \times r$ matrices: special form, e.g., Toeplitz, Hankel, Vandermonde, Cauchy, Sylvester, etc) $\longrightarrow$ encoded by $O(r)$ elements
  - for polynomials/power series: satisfying an equation (algebraic or differential) $\longrightarrow$ encoded by degree/order of size $O(1)$

- In this lecture, we focus on dense objects

Typical problems

- On all objects
  - sum, product
  - inversion, division
- On power series
  - logarithm, exponential
  - composition
  - Padé and Hermite-Padé approximation
- On polynomials
  - (multipoint) evaluation, interpolation
  - (extended) greatest commun divisor, resultant
  - shift
  - composed sum and product
- On matrices
  - system solving
  - determinant, characteristic polynomial

Complexity yardsticks

Important issues:

- addition is easy: naive algorithm already optimal
- multiplication is the most basic (non-trivial) problem
- almost all problems can be reduced to multiplication

Are there quasi-optimal algorithms for:

- polynomial/power series multiplication? Yes!
- matrix multiplication? Big open problem!

Complexity yardsticks

$$\begin{aligned}
\mathsf{M}(n) &= \text{complexity of polynomial multiplication in } \mathbb{K}[x]_{<n} \\
&= O(n^2) \text{ by the naive algorithm} \\
&= O\big(n^{1.58}\big) \text{ by Karatsuba's algorithm} \\
&= O\big(n^{\log_\alpha (2\alpha-1)}\big) \text{ by the Toom-Cook algorithm } (\alpha \geq 3) \\
&= O\big(n \log n \operatorname{loglog} n\big) \text{ by the Schönhage-Strassen algorithm}
\end{aligned}$$

$$\begin{aligned}
\mathsf{MM}(r) &= \text{complexity of matrix product in } \mathcal{M}_r(\mathbb{K}) \\
&= O(r^3) \text{ by the naive algorithm} \\
&= O(r^{2.81}) \text{ by Strassen's algorithm} \\
&= O(r^{2.38}) \text{ by the Coppersmith-Winograd algorithm}
\end{aligned}$$

$$\begin{aligned}
\mathsf{MM}(r,n) &= \text{complexity of polynomial matrix product in } \mathcal{M}_r(\mathbb{K}[x]_{<n}) \\
&= O(r^3\, \mathsf{M}(n)) \text{ by the naive algorithm} \\
&= O(\mathsf{MM}(r)\, n \log(n) + r^2 n \log n \operatorname{loglog} n) \text{ by the Cantor-Kaltofen algo} \\
&= O(\mathsf{MM}(r)\, n + r^2\, \mathsf{M}(n)) \text{ by the B-Schost algorithm}
\end{aligned}$$

Typical problems, and their complexities

• Polynomials, power series and matrices	
– product	$\mathsf{M}(n)$, $\mathsf{MM}(r)$
– division/inversion	$O(\mathsf{M}(n))$, $O(\mathsf{MM}(r))$
• On power series	
– logarithm, exponential	$O(\mathsf{M}(n))$
– composition	$O(\sqrt{n \log n}\,\mathsf{M}(n))$
– Padé approximation	$O(\mathsf{M}(n) \log n)$
• On polynomials	
– (multipoint) evaluation, interpolation	$O(\mathsf{M}(n) \log n)$
– extended gcd, resultant	$O(\mathsf{M}(n) \log n)$
– shift	$O(\mathsf{M}(n))$
– composed sum and product	$O(\mathsf{M}(n))$
• On matrices	
– system solving, determinant	$O(\mathsf{MM}(r))$
– characteristic / minimal polynomial	$O(\mathsf{MM}(r))$

Typical problems, and the algorithms' designers

- Polynomials, power series and matrices
  - product
  - division/inversion — Sieveking-Kung 1972, Strassen 1969, 1973
- On power series
  - logarithm, exponential — Brent 1975
  - composition — Brent-Kung 1978
  - Padé approximation — Brent-Gustavson-Yun 1980
- On polynomials
  - (multipoint) evaluation, interpolation — Borodin-Moenck 1974
  - extended gcd, resultant — Knuth-Schönhage 1971, Schwartz 1980
  - shift — Aho-Steiglitz-Ullman 1975
  - composed sum and product — B-Flajolet-Salvy-Schost 2006
- On matrices
  - system solving, determinant — Strassen 1969
  - characteristic polynomial / minimal polynomial — Keller-Gehrig 1985

Typical problems, and their complexities

- On power series matrices
  - product $\mathsf{MM}(r,n)$
  - inversion $O(\mathsf{MM}(r,n))$
  - quasi-exponential (sol. of $Y' = AY$) $O(\mathsf{MM}(r,n))$
- On power series
  - Hermite-Padé approximation of r series $O(\mathsf{MM}(r,n)\log n)$
- On polynomial matrices
  - product $\mathsf{MM}(r,n)$
  - system solving $O(\mathsf{MM}(r,n)\log n)$
  - determinant $O(\mathsf{MM}(r,n)\log^2(n))$
  - inversion $\tilde{O}(r^3\, n)$, if $r = 2^k$
  - characteristic / minimal polynomial $\tilde{O}(r^{2.6972}\, n)$

Typical problems, and the algorithms' designers

• **On power series matrices**	
– product	
– inversion	Schulz 1933
– quasi-exponential	B-Chyzak-Ollivier-Salvy-Schost-Sedoglavic 2007
• **On power series**	
– Hermite-Padé approximation	Beckermann-Labahn 1994
• **On polynomial matrices**	
– product	
– system solving	Storjohann 2002
– determinant	Storjohann 2002
– inversion	Jeannerod-Villard 2005
– characteristic / minimal polynomial	Kaltofen-Villard 2004

Other problems, and their complexities

- On structured (D-finite, algebraic) power series
  - sum, product, Hadamard product $O(n)$
  - inversion $O(\mathsf{M}(n))$, $O(n)$
- On structured matrices
  - Toeplitz-like: system solving, determinant $O(\mathsf{M}(r)\log r)$
  - Vandermonde-like: system solving, determinant $O(\mathsf{M}(r)\log^2(r))$
  - Cauchy-like: system solving, determinant $O(\mathsf{M}(r)\log^2(r))$
- On sparse matrices
  - system solving $O(r^2)$
  - determinant $O(r^2)$
  - rank $O(r^2)$
  - minimal polynomial $O(r^2)$

Other problems, and their complexities

- On structured (D-finite, algebraic) power series
  - sum, product, Hadamard product — folklore, but not sufficiently known!
  - inversion
- On structured matrices
  - Toeplitz-like: system solving, determinant — Bitmead-Anderson-Morf 1980
  - Vandermonde-like: system solving, determinant — Pan 1990
  - Cauchy-like: system solving, determinant — Pan 2000
- On sparse matrices
  - system solving — Wiedemann 1986
  - determinant — Wiedemann 1986
  - rank — Kaltofen-Saunders 1991
  - minimal polynomial — Wiedemann 1986

Algorithmic paradigms

Given a problem, how to find an efficient algorithm for its solution?

Five paradigms for algorithmic design

- divide and conquer (DAC)
- decrease and conquer (dac)
- baby steps / giant steps (BS-GS)
- change of representation (CR)
- Tellegen's transposition principle (Tellegen)

Algorithmic paradigms, and main techniques

Given a problem, how to find an efficient algorithm for its solution?

Five paradigms for algorithmic design

- divide and conquer
- decrease and conquer
  - binary powering
  - Newton iteration
  - Keller-Gehrig iteration
- baby steps / giant steps
- change of representation
  - evaluation-interpolation
  - expansion-reconstruction
- Tellegen's transposition principle

Divide and conquer

Idea: recursively break down a problem into two or more similar subproblems, solve them, and combine their solutions

Origin: unknown, probably very ancient.
Modern form: merge sort algorithm — von Neumann 1945

Our main examples:

• Karatsuba algorithm	polynomial multiplication
• Strassen algorithm	matrix product
• Strassen algorithm	matrix inversion
• Borodin-Moenck algorithm	polynomial evaluation-interpolation
• Beckermann-Labahn algorithm	Hermite-Padé approximation
• Bitmead-Anderson-Morf algorithm	solving Toeplitz-like linear systems
• Lehmer-Knuth-Schönhage-Moenck-Strassen algorithm	extended gcd

Decrease and conquer

Idea: reduce each problem to only one similar subproblem of half size

Origin: probably Pingala's Hindu classic Chandah-sutra, 200 BC

Modern form: binary search algorithm Mauchly 1946

Our main examples:

- binary powering exponentiation in rings
- modular exponentiation exponentiation in quotient rings
  - N-th term of a recurrence with constant coefficients
- Newton iteration power series root-finding
  - polynomial division
  - composed sum and product
  - polynomial shift
- Kehler-Gehrig algorithm Krylov sequence computation
- Storjohann's high order lifting algorithm polynomial matrices
- B-Schost algorithm interpolation on geometric sequences

Baby steps / giant steps

Idea: reduce a problem of size N to two similar subproblem of size $\sqrt{N}$

Origin: computational number theory, ≈ 1960

Modern form: discrete logarithm problem — Shanks 1969

Our main examples:

- Paterson-Stockmeyer 1973 — polynomial evaluation in an algebra
- Strassen 1976 — deterministic integer factorization
- Brent-Kung 1978 — composition of power series
- Chudnovsky-Chudnovsky 1987 — N-th term of a P-recursive sequence
  - point counting on hyperelliptic curves
  - polynomial solutions of linear differential equations
  - p-curvature of linear differential operators
- Shoup 1995 — power projection $[\ell(1), \ell(u), \ldots, \ell(u^{N-1})]$

Change of representation

Idea: represent objects in a different way, mathematically equivalent, but better suited for the algorithmic treatment

Origin: unknown, probably Sun Zi ≈ 300 (Chinese remainder theorem)
Modern form: the Czech number system Svoboda-Valach 1955

Our main examples: One can represent

- a polynomial by
  - the list of its coefficients
  - the values it takes at sufficiently many points easy $\times$
  - its Newton sums (= powersums of roots) easy $\otimes, \oplus$
- a rational fraction by
  - the coefficient lists of its denominator and numerator
  - its values at sufficiently many points
  - its Taylor series expansion

Tellegen's transposition principle

Idea: to solve a linear problem, find an algorithm for its dual, and transpose it

Origin: electrical network theory: Tellegen, Bordewijk, ≈ 1950
Modern form: transposition of algorithms, complexity version Fiduccia 1972

Our main examples:

- improve algorithms by constant factors
  - Hanrot-Quercia-Zimmermann 2002 middle product for polynomials
  - B-Lecerf-Schost 2003 multipoint evaluation and interpolation
- prove computational equivalence between problems
  - B-Schost 2004 multipoint evaluation $\Leftrightarrow$ interpolation
- discover new algorithms
  - B-Salvy-Schost 2008 base conversions
- understand (connections between) existing algorithms
  - DFT: decimation in time vs. decimation in frequency
  - Strassen's polynomial division vs. Shoup's extension of recurrences

The Master Theorem

Suppose that the complexity $\mathsf{C}(n)$ of an algorithm satisfies

$$\mathsf{C}(n) \leq s \cdot \mathsf{C}\left(\frac{n}{2}\right) + \mathsf{T}(n),$$

where the function T is such that $q\mathsf{T}(n) \leq \mathsf{T}(2n)$. Then, for $n \to \infty$

$$\mathsf{C}(n) \; = \begin{cases} O(\mathsf{T}(n)), & \text{if } s < q, \\ O(\mathsf{T}(n) \log n), & \text{if } s = q, \\ O\left(n^{\log s} \frac{\mathsf{T}(n)}{n^{\log q}}\right), & \text{if } s > q. \end{cases}$$

Proof:

$$\begin{aligned} \mathsf{C}(n) &\leq \mathsf{T}(n) + s \cdot \mathsf{C}\left(\frac{n}{2}\right) \\ &\leq \mathsf{T}(n) + s \cdot \mathsf{T}\left(\frac{n}{2}\right) + \cdots + s^{k-1} \cdot \mathsf{T}\left(\frac{n}{2^{k-1}}\right) + s^k \cdot \mathsf{C}\left(\frac{n}{2^k}\right) \\ &\leq \mathsf{T}(n) \cdot \left(1 + \frac{s}{q} + \cdots + \left(\frac{s}{q}\right)^{\log(n)-1}\right) + s^{\log n} \cdot \mathsf{C}(1) \end{aligned}$$

The Master Theorem, main consequences

Corollary DFT / Karatsuba

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(n) \qquad \Longrightarrow \qquad \mathsf{C}(n) = \begin{cases} O(n \log n), & \text{if } s = 2 \\ O(n^{\log s}), & \text{if } s \ge 3 \end{cases}$$

Corollary Newton / evaluation-interpolation

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(\mathsf{M}(n)) \qquad \Longrightarrow \qquad \mathsf{C}(n) = \begin{cases} O(\mathsf{M}(n)), & \text{if } s = 1 \\ O(\mathsf{M}(n) \log n), & \text{if } s = 2 \end{cases}$$

Corollary Strassen's matrix product

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(n^2), \quad (s \ge 5) \qquad \Longrightarrow \qquad \mathsf{C}(n) = O(n^{\log s})$$

Corollary Strassen's matrix inversion

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(\mathsf{MM}(n)), \quad (s \le 3) \qquad \Longrightarrow \qquad \mathsf{C}(n) = O(\mathsf{MM}(n))$$

Divide and conquer

Karatsuba's algorithm

Gauss's trick ($\approx$1800) The product of two complex numbers can be computed using only 3 real multiplications

$$(ai+b)(ci+d) = (ad+bc)i + (bd-ac) = ((a+b)(c+d) - bd - ac)i + (bd-ac)$$

Kolmogorov (1956) n^2 conjecture: n^2 ops. are needed to multiply n-digit integers

Karatsuba (1960) disproof of the Kolmogorov conjecture

$\longrightarrow$ first DAC algorithm in Computer algebra; it combines Gauss's trick (on polynomials) with the power of recursion

$$(ax^{n/2}+b)(cx^{n/2}+d) = acx^n + ((a+b)(c+d) - bd - ac)x^{n/2} + bd$$

Master Theorem: $\mathsf{K}(n) = 3\cdot\mathsf{K}(n/2) + O(n) \implies \mathsf{K}(n) = O(n^{\log(3)}) = O(n^{1.59})$

The idea behind the trick

Let $f = ax + b$, $g = cx + d$. Compute $h = fg$ by evaluation-interpolation:

Evaluation:

$$\begin{aligned} b &= f(0) & \qquad d &= g(0) \\ a+b &= f(1) & \qquad c+d &= g(1) \\ a &= f(\infty) & \qquad c &= g(\infty) \end{aligned}$$

Multiplication:

$$\begin{aligned} h(0) &= f(0)\cdot g(0) \\ h(1) &= f(1)\cdot g(1) \\ h(\infty) &= f(\infty)\cdot g(\infty) \end{aligned}$$

Interpolation:

$$h = h(0) + (h(1) - h(0) - h(\infty))\,x + h(\infty)\,x^2$$

Toom's algorithm

Let

$$f = f_0 + f_1 x + f_2 x^2, \quad g = g_0 + g_1 x + g_2 x^2$$

and

$$h = fg = h_0 + h_1 x + h_2 x^2 + h_3 x^3 + h_4 x^4.$$

To get h, do again:

- evaluation,
- multiplication,
- interpolation.

Now, 5 values are needed, because h has 5 unknown coefficients:

- $0, 1, -1, 2, \infty$ other choices are possible
- would not work with coefficients in $\mathbb{F}_2$.

The evaluation / interpolation phase

Evaluation:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
f(0) & = & f_0 & g(0) & = & g_0 \\
f(1) & = & f_0 + f_1 + f_2 & g(1) & = & g_0 + g_1 + g_2 \\
f(-1) & = & f_0 - f_1 + f_2 & g(-1) & = & g_0 - g_1 + g_2 \\
f(2) & = & f_0 + 2f_1 + 4f_2 & g(2) & = & g_0 + 2g_1 + 4g_2 \\
f(\infty) & = & f_2 & g(\infty) & = & g_2
\end{array}$$

Multiplication:

$$h(0) = f(0)g(0), \quad \ldots, \quad h(\infty) = f(\infty)g(\infty)$$

Interpolation: recover h from its values.

$\Longrightarrow$ one can multiply degree-2 polynomials using 5 products instead of 9

Master Theorem: $\mathsf{T}(n) = 5 \cdot \mathsf{T}(n/3) + O(n) \implies \mathsf{T}(n) = O(n^{\log_3(5)}) = O(n^{1.47})$

Generalization of Toom

Let

$$f = f_0 + f_1 x + \cdots + f_{\alpha-1} x^{\alpha-1}, \quad g = g_0 + g_1 x + \cdots + g_{\alpha-1} x^{\alpha-1}$$

and

$$h = fg = h_0 + h_1 x + \cdots + h_{2\alpha-2} x^{2\alpha-2}.$$

Analysis: at each step,

- divide n by α; number of terms in f, g
- and perform $2\alpha - 1$ recursive calls; number of terms in h
- the extra operations count is ℓn, for some ℓ.

Master theorem:

$$\mathsf{T}(n) = O(n^{\log_\alpha(2\alpha-1)}).$$

Examples:

$$\alpha = 100 \implies O(n^{1.15}), \quad \alpha = 1000 \implies O(n^{1.1}), \quad \alpha = 10000 \implies O(n^{1.07})$$

Discrete Fourier Transform (Gentleman-Sande 1966, decimation-in-frequency)

Problem: Given $n = 2^k$, $f \in \mathbb{K}[x]_{<n}$, and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$

Idea: $\omega = n$-th primitive root of 1 $\Longrightarrow \omega^2 = \frac{n}{2}$-th primitive root of 1, and

$$\begin{aligned} r_0(x) =& f(x) \bmod x^{n/2} - 1 && \Longrightarrow && f(\omega^{2j}) = r_0\left((\omega^2)^j\right) \\ r_1(x) =& f(x) \bmod x^{n/2} + 1 && \Longrightarrow && f(\omega^{2j+1}) = r_1(\omega^{2j+1}) = r_1(\omega x)\big|_{x=(\omega^2)^j} \end{aligned}$$

Moreover, $O(n)$ ops. are enough to get $r_0(x), r_1(x), r_1(\omega x)$ from $f(x)$

Master Theorem: $\mathsf{F}(n) = 2 \cdot \mathsf{F}(n/2) + O(n) \quad \Longrightarrow \quad \mathsf{F}(n) = O(n \log n)$

Discrete Fourier Transform (Cooley-Tukey 1965, decimation-in-frequency)

Problem: Given $n = 2^k$, $f \in \mathbb{K}[x]_{<n}$, and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$

Idea: Write $f = f_{\text{even}}(x^2) + x f_{\text{odd}}(x^2), \quad$ with $\deg(f_{\text{even}}), \deg(f_{\text{odd}}) < n/2$.

Then $f(\omega^j) = f_{\text{even}}(\omega^{2j}) + \omega^j f_{\text{odd}}(\omega^{2j})$, and $(\omega^{2j})_{0\le j<n} = \frac{n}{2}$-roots of 1.

Master Theorem: $\quad \mathsf{F}(n) = 2 \cdot \mathsf{F}(n/2) + O(n) \quad \Longrightarrow \quad \mathsf{F}(n) = O(n \log n)$

Inverse DFT

Problem: Given $n = 2^k$, $v_0, \ldots, v_{n-1} \in \mathbb{K}$ and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $f \in \mathbb{K}[x]_{<n}$ such that $f(1) = v_0, \ldots, f(\omega^{n-1}) = v_{n-1}$

- $V_\omega \cdot V_{\omega^{-1}} = n \cdot I_n \;\rightarrow\;$ performing the inverse DFT in size n amounts to:
  - performing a DFT at
$$\frac{1}{1}, \;\; \frac{1}{\omega}, \;\; \ldots, \;\; \frac{1}{\omega^{n-1}}$$
  - dividing the results by n.
- this new DFT is the same as before:
$$\frac{1}{\omega^i} = \omega^{n-i},$$
  so the outputs are just shuffled.

Consequence: the cost of the inverse DFT is $O(n \log(n))$

FFT polynomial multiplication

Suppose the basefield $\mathbb{K}$ contains enough roots of unity

To multiply two polynomials f, g in $\mathbb{K}[x]$, of degrees $< n$:

- find $N = 2^k$ such that $h = fg$ has degree less than N $\quad N \leq 4n$
- compute DFT(f, N) and DFT(g, N) $\quad O(N \log(N))$
- multiply the values to get DFT(h, N) $\quad O(N)$
- recover h by inverse DFT $\quad O(N \log(N))$

Cost: $O(N \log(N)) = O(n \log(n))$

General case: Create artificial roots of unity $\quad O(n \log(n) \log \log n)$

Strassen's matrix multiplication algorithm

Same idea as for Karatsuba's algorithm: trick in low size + recursion

Additional difficulty: Formulas should be non-commutative

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & t \end{bmatrix} \iff \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \times \begin{bmatrix} x \\ z \\ y \\ t \end{bmatrix}$$

Crucial remark: If $\varepsilon \in \{0,1\}$ and $\alpha \in \mathbb{K}$, then 1 multiplication suffices for $E \cdot v$, where v is a vector, and E is a matrix of one of the following types:

$$\begin{bmatrix} \alpha & \alpha \\ \varepsilon\alpha & \varepsilon\alpha \end{bmatrix}, \begin{bmatrix} \alpha & -\alpha \\ \varepsilon\alpha & -\varepsilon\alpha \end{bmatrix}, \begin{bmatrix} \alpha & \varepsilon\alpha \\ -\alpha & -\varepsilon\alpha \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - \underbrace{\begin{bmatrix} a & a & & \\ a & a & & \\ & & & \\ & & & \end{bmatrix}}_{E_1} - \underbrace{\begin{bmatrix} & & & \\ & & & \\ & & d & d \\ & & d & d \end{bmatrix}}_{E_2} = \begin{bmatrix} & b-a & & \\ c-a & d-a & & \\ & & a-d & b-d \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 = \underbrace{\begin{bmatrix} & & & \\ & d-a & a-d & \\ & d-a & a-d & \\ & & & \end{bmatrix}}_{E_3} - \begin{bmatrix} & b-a & & \\ c-a & & d-a & \\ & a-d & & b-d \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 - E_3 = \begin{bmatrix} & b-a & & \\ & & & \\ & a-d & & b-d \\ & & & \end{bmatrix} + \begin{bmatrix} & & & \\ c-a & & d-a & \\ & & & \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 - E_3 = \underbrace{\begin{bmatrix} & b-a & & \\ & & & \\ & (b-d)-(b-a) & & b-d \\ & & & \end{bmatrix}}_{E_4 \quad + \quad E_5} + \underbrace{\begin{bmatrix} & & & \\ c-a & & (c-a)-(c-d) & \\ & & & \\ & & c-d & \end{bmatrix}}_{E_6 \quad + \quad E_7}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

Conclusion

$$M = E_1 + E_2 + E_3 + E_4 + E_5 + E_6 + E_7$$

$\implies$ one can multiply 2×2 matrices using 7 products instead of 8

Master Theorem:

$\mathsf{MM}(r) = 7 \cdot \mathsf{MM}(r/2) + O(r^2) \quad \implies \quad \mathsf{MM}(r) = O(r^{\log_2(7)}) = O(r^{2.81})$

Inversion of dense matrices

[Strassen, 1969]

To invert a dense matrix $A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \in \mathcal{M}_r(\mathbb{K})$:

1. Invert $A_{1,1}$ (recursively)
2. Compute the Schur complement $\Delta := A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{1,2}$
3. Invert Δ (recursively)
4. Recover the inverse of A as

$$A^{-1} = \begin{bmatrix} I & -A_{1,1}^{-1} A_{1,2} \\ & I \end{bmatrix} \times \begin{bmatrix} A_{1,1}^{-1} & \\ & \Delta^{-1} \end{bmatrix} \times \begin{bmatrix} I & \\ -A_{2,1} A_{1,1}^{-1} & I \end{bmatrix}$$

Master Theorem: $\mathsf{C}(r) = 2 \cdot \mathsf{C}\left(\frac{r}{2}\right) + O(\mathsf{MM}(r)) \quad \Longrightarrow \quad \mathsf{C}(r) = O(\mathsf{MM}(r))$

Corollary: inversion A^{-1} and system solving $A^{-1}b$ in time $O(\mathsf{MM}(r))$

Subproduct tree

Problem: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$, compute $A = \prod_{i=0}^{n-1}(x - a_i)$

$$\begin{tikzcd}
 & & & & \boxed{A = \prod_{i=0}^{n-1}(x-a_i)} & & & & \\
 & & \boxed{B_0 = \prod_{i=0}^{n/2-1}(x-a_i)} \arrow[urr] & & & & \boxed{B_1 = \prod_{i=n/2}^{n-1}(x-a_i)} \arrow[ull] & & \\
 & \vdots \arrow[ur] & & \vdots \arrow[ul] & & \vdots \arrow[ur] & & \vdots \arrow[ul] & \\
 & \boxed{(x-a_0)(x-a_1)} & & & & & & \boxed{(x-a_{n-2})(x-a_{n-1})} & \\
\boxed{x-a_0} \arrow[ur] & & \boxed{x-a_1} \arrow[ul] & & \cdots & & \boxed{x-a_{n-2}} \arrow[ur] & & \boxed{x-a_{n-1}} \arrow[ul]
\end{tikzcd}$$

Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \quad \Longrightarrow \quad \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$

Fast multipoint evaluation

[Borodin-Moenck, 1974]

Pb: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$ and $P \in \mathbb{K}[x]_{<n}$, compute $P(a_0), \dots, P(a_{n-1})$

Naive algorithm: Compute $P(a_i)$ independently $O(n^2)$

Basic idea: Use recursively Bézout's identity $P(a) = P(x) \bmod (x - a)$

Divide and conquer: Same idea as for DFT = evaluation by repeated division

- $P_0 = P \mod (x - a_0) \cdots (x - a_{n/2-1})$
- $P_1 = P \mod (x - a_{n/2}) \cdots (x - a_{n-1})$

$$\Longrightarrow \begin{cases} P_0(a_0) = P(a_0), \quad \dots, \quad P_0(a_{n/2-1}) = P(a_{n/2-1}) \\ P_1(a_{n/2}) = P(a_{n/2}), \quad \dots, \quad P_1(a_{n-1}) = P(a_{n-1}) \end{cases}$$

Fast multipoint evaluation

[Borodin-Moenck, 1974]

Pb: Given $a_0, \ldots, a_{n-1} \in \mathbb{K}$ and $P \in \mathbb{K}[x]_{<n}$, compute $P(a_0), \ldots, P(a_{n-1})$

$$\begin{tikzcd}
 & & & \boxed{P} \arrow[dll, "\mathrm{mod}(\cdot, B_0)"'] \arrow[drr, "\mathrm{mod}(\cdot, B_1)"] & & & \\
 & \boxed{P \bmod B_0} \arrow[dl, "\mathrm{mod}"'] \arrow[dr, "\mathrm{mod}"] & & & & \boxed{P \bmod B_1} \arrow[dl, "\mathrm{mod}"'] \arrow[dr, "\mathrm{mod}"] & \\
\vdots & & \vdots & & \vdots & & \vdots \\
 & \boxed{P \bmod (x-a_0)(x-a_1)} \arrow[dl, "\mathrm{mod}"'] \arrow[dr, "\mathrm{mod}"] & & & & \boxed{P \bmod (x-a_{n-2})(x-a_{n-1})} \arrow[dl, "\mathrm{mod}"'] \arrow[dr, "\mathrm{mod}"] & \\
\boxed{P \bmod (x-a_0)} & & \boxed{P \bmod (x-a_1)} & \cdots & \boxed{P \bmod (x-a_{n-2})} & & \boxed{P \bmod (x-a_{n-1})}
\end{tikzcd}$$

Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \implies \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$

Fast interpolation

[Borodin-Moenck, 1974]

Problem: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$ mutually distinct, and $v_0, \dots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(a_0) = v_0, \dots, P(a_{n-1}) = v_{n-1}$

Naive algorithm: Linear algebra, Vandermonde system $O(\mathsf{MM}(n))$

Lagrange's algorithm: Use $P(x) = \sum_{i=0}^{n-1} v_i \frac{\prod_{j \neq i}(x - a_j)}{\prod_{j \neq i}(a_i - a_j)}$ $O(n^2)$

Fast algorithm: Modified Lagrange formula

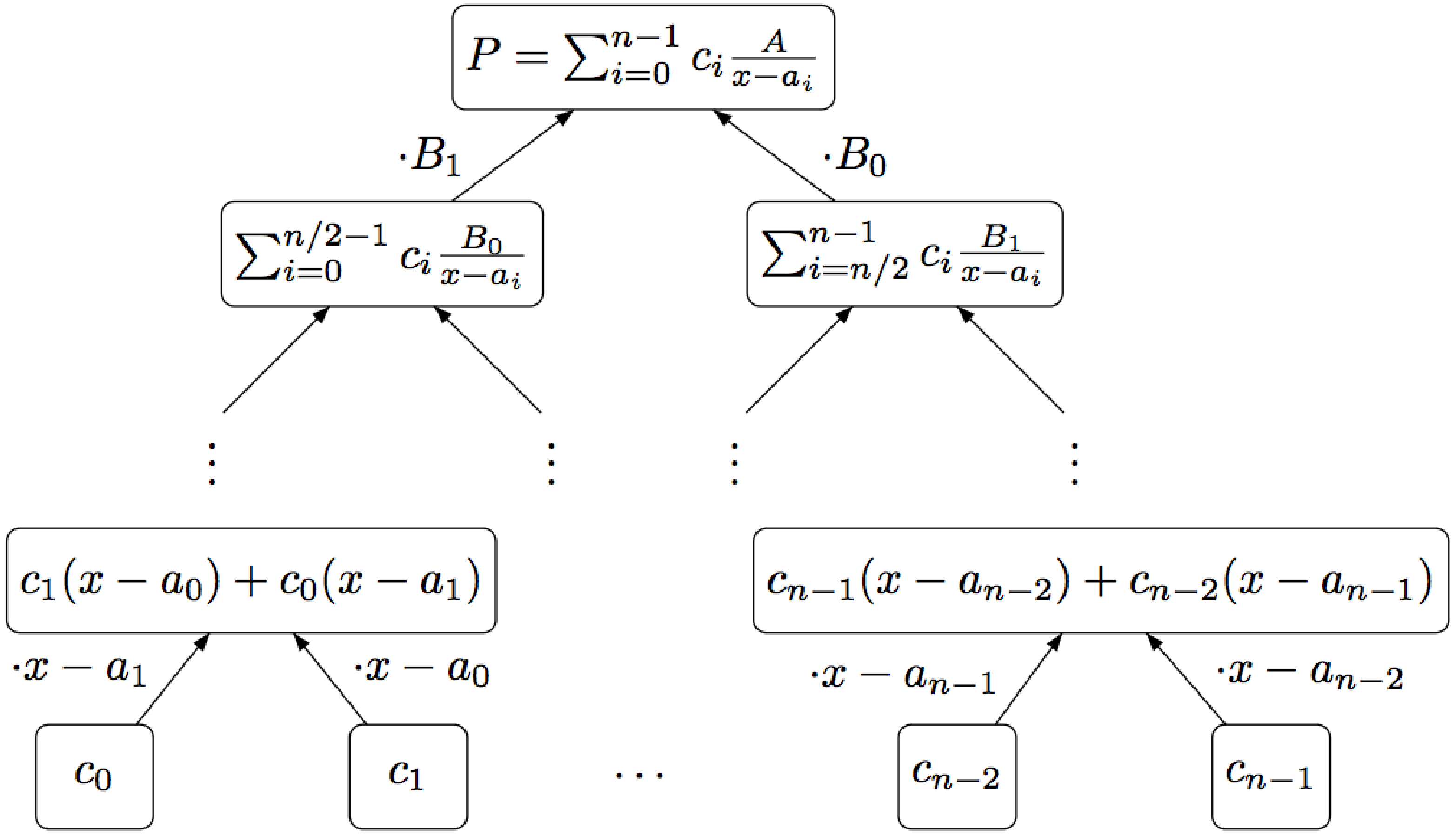
$$P = A(x) \cdot \sum_{i=0}^{n-1} \frac{v_i / A'(a_i)}{x - a_i}$$

- Compute $c_i = v_i / A'(a_i)$ by fast multipoint evaluation $O(\mathsf{M}(n) \log n)$
- Compute $\sum_{i=0}^{n-1} \frac{c_i}{x - a_i}$ by divide and conquer $O(\mathsf{M}(n) \log n)$

Fast interpolation

[Borodin-Moenck, 1974]

Problem: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$ mutually distinct, and $v_0, \dots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(a_0) = v_0, \dots, P(a_{n-1}) = v_{n-1}$



$P = \sum_{i=0}^{n-1} c_i \frac{A}{x-a_i}$
$\cdot B_1$
$\cdot B_0$
$\sum_{i=0}^{n/2-1} c_i \frac{B_0}{x-a_i}$
$\sum_{i=n/2}^{n-1} c_i \frac{B_1}{x-a_i}$
$c_1(x-a_0) + c_0(x-a_1)$
$c_{n-1}(x-a_{n-2}) + c_{n-2}(x-a_{n-1})$
$\cdot x - a_1$
$\cdot x - a_0$
$\cdot x - a_{n-1}$
$\cdot x - a_{n-2}$
c_0
c_1
$\cdots$
c_{n-2}
c_{n-1}

Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \implies \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$

Decrease and conquer I

Evaluation-interpolation, geometric case

Subproduct tree, geometric case

[B-Schost, 2005]

Problem: Given $q \in \mathbb{K}$, compute $A = \prod_{i=0}^{n-1}(x - q^i)$

Idea: Compute $B_1 = \prod_{i=n/2}^{n-1} (x - q^i)$ from $B_0 = \prod_{i=0}^{n/2-1} (x - q^i)$, by a homothety

$$B_1(x) = B_0\left(\frac{x}{q^{n/2}}\right) \cdot q^{(n/2)^2}$$

Decrease and conquer:

- Compute $B_0(x)$ by a recursive call
- Deduce $B_1(x)$ from $B_0(x)$ $\quad O(n)$
- Return $A(x) = B_0(x)B_1(x)$ $\quad \mathsf{M}(n/2)$

Master Theorem: $\mathsf{C}(n) = \mathsf{C}(n/2) + O(\mathsf{M}(n)) \quad \Longrightarrow \quad \mathsf{C}(n) = O(\mathsf{M}(n))$

Fast multipoint evaluation, geometric case

[Bluestein, 1970]

Problem: Given $q \in \mathbb{K}$ and $P \in \mathbb{K}[x]_{<n}$, compute $P(1), P(q), \ldots, P(q^{n-1})$

The needed values are: $$P(q^i) = \sum_{j=0}^{n-1} c_j q^{ij}, \qquad 0 \le i < n$$

Bluestein's trick: $$ij = \frac{(i+j)^2 - i^2 - j^2}{2} \quad \Longrightarrow \quad q^{ij} = q^{(i+j)^2/2} \cdot q^{-i^2/2} \cdot q^{-j^2/2}$$

$$\Longrightarrow \qquad P(q^i) = q^{-i^2/2} \cdot \underbrace{\sum_{j=0}^{n-1} c_j q^{-j^2/2} \cdot q^{(i+j)^2/2}}_{\text{convolution:}}$$

$$\left[x^{n-1+i}\right] \left(\sum_{k=0}^{n-1} c_k q^{-k^2/2} x^{n-k-1} \right) \left(\sum_{\ell=0}^{2n-2} q^{\ell^2/2} x^{\ell} \right)$$

Conclusion: Fast evaluation on a geometric sequence in $O(\mathsf{M}(n))$

Fast interpolation, geometric case

[B-Schost, 2005]

Problem: Given $q \in \mathbb{K}$, and $v_0, \dots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(1) = v_0, \dots, P(q^{n-1}) = v_{n-1}$

Fast algorithm: Modified Lagrange formula

$$P = A(x) \cdot \sum_{i=0}^{n-1} \frac{v_i / A'(q^i)}{x - q^i}, \qquad A = \prod_i (x - q^i)$$

- Compute $\prod_{i=0}^{n-1} (x - q^i)$ by decrease and conquer $O(\mathsf{M}(n))$
- Compute $c_i = v_i / A'(q^i)$ by Bluestein's algorithm $O(\mathsf{M}(n))$
- Compute $\sum_{i=0}^{n-1} \frac{c_i}{x - q^i}$ by decrease and conquer $O(\mathsf{M}(n))$

Fast interpolation, geometric case

[B-Schost, 2005]

Problem: Given $q \in \mathbb{K}$, and $v_0, \ldots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(1) = v_0, \ldots, P(q^{n-1}) = v_{n-1}$

Subproblem: Given $c_0, \ldots, c_{n-1} \in \mathbb{K}$, compute $\quad R(x) = \sum_{i=0}^{n-1} \frac{c_i}{x - q^i}$

Idea: change of representation – enough to compute $R \mod x^n$

Second idea: $R \bmod x^n$ = multipoint evaluation at $\{1, q^{-1}, \ldots, q^{-(n-1)}\}$:

$$\sum_{i=0}^{n-1} \frac{c_i}{x - q^i} \mod x^n = -\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} c_i q^{-i(j+1)} x^j \right) = -\sum_{j=0}^{n-1} C(q^{-j-1}) x^j$$

Conclusion: Algorithm for interpolation at a geometric sequence in $O(\mathsf{M}(n))$ (generalization of the IDFT)

Product of polynomial matrices

[B-Schost, 2005]

Problem: Given $A, B \in \mathcal{M}_r(\mathbb{K}[x]_{<n})$, compute $C = AB$

Idea: change of representation – evaluation-interpolation at a geometric sequence $\mathcal{G} = \{1, q, q^2, \ldots, q^{2n-2}\}$

- Evaluate A and B at $\mathcal{G}$ $\quad O(r^2\,\mathsf{M}(n))$
- Multiply values $C(v) = A(v)B(v)$ for $v \in \mathcal{G}$ $\quad O(n\,\mathsf{MM}(r))$
- Interpolate C from values $\quad O(r^2\,\mathsf{M}(n))$

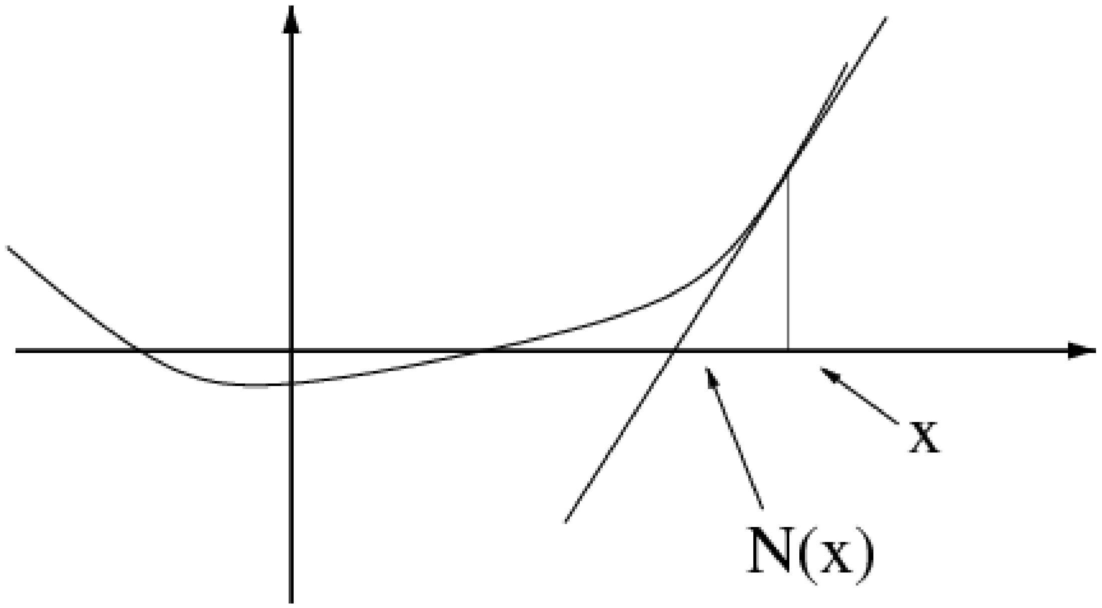
Total complexity $\quad O(r^2\,\mathsf{M}(n) + n\,\mathsf{MM}(r))$

Decrease and conquer II

Newton iteration

Newton's tangent method: real case

[Newton, 1671]



$$\mathbf{x_{\kappa+1} = N(x_\kappa) = x_\kappa - (x_\kappa^2 - 2)/(2x_\kappa), \quad x_0 = 1.5}$$

$$\mathbf{x_1 = 1.41666666666666666666666666666667}$$

$$\mathbf{x_2 = 1.4142156862745098039215686274510}$$

$$\mathbf{x_3 = 1.41421356237468991062629557889013}$$

$$\mathbf{x_4 = 1.41421356237309504880168962350253}$$

Newton's tangent method: power series case

Let $\varphi : \mathbb{K}[[x]] \to \mathbb{K}[[x]]$. To solve $\varphi(g) = 0$ in $\mathbb{K}[[x]]$, iterate

$$g_{\kappa+1} = g_\kappa - \frac{\varphi(g_\kappa)}{\varphi'(g_\kappa)} \mod x^{2^{\kappa+1}}$$

- The number of correct coefficients doubles after each iteration
- Total cost $\quad = \quad \mathbf{2} \times \Big($ the cost of the last iteration $\Big)$

Theorem [Cook (1966), Sieveking (1972) & Kung (1974), Brent (1975)]
Division, logarithm and exponential of power series in $\mathbb{K}[[x]]$ *can be computed at precision* $\mathbf{N}$ *using* $\mathbf{O}(\mathsf{M}(\mathbf{N}))$ operations in $\mathbb{K}$

Division and logarithm of power series

[Sieveking-Kung, 1972]

To compute the reciprocal of $f \in \mathbb{K}[[x]]$, choose $\varphi(g) = 1/g - f$:

$$g_0 = \frac{1}{f_0} \quad \text{and} \quad g_{\kappa+1} = g_\kappa + g_\kappa(1 - f g_\kappa) \quad \text{mod } x^{2^{\kappa+1}} \qquad \text{for } \kappa \geq 0$$

Master Theorem: $\mathsf{C}(N) = \mathsf{C}(N/2) + O(\mathsf{M}(N)) \qquad \Longrightarrow \qquad \mathsf{C}(N) = O(\mathsf{M}(N))$

Corollary: division of power series at precision N in $O(\mathsf{M}(N))$

Corollary: Logarithm $\log(f) = -\sum_{i\geq 1} \frac{(1-f)^i}{i}$ of $f \in 1 + x\mathbb{K}[[x]]$ in $O(\mathsf{M}(N))$:

- compute the Taylor expansion of $h = f'/f$ modulo x^{N-1} $\qquad O(\mathsf{M}(N))$
- take the antiderivative of h $\qquad O(N)$

Application: polynomial division

[Strassen, 1973]

Pb: Given $F, G \in \mathbb{K}[x]_{\leq N}$, compute (Q, R) in Euclidean division $F = QG + R$

Naive algorithm: $O(N^2)$

Idea: look at $F = QG + R$ from the infinity: $Q \sim_{+\infty} F/G$

Let $N = \deg(F)$ and $n = \deg(G)$. Then $\deg(Q) = N - n$, $\deg(R) < n$ and

$$\underbrace{F(1/x)x^N}_{\mathrm{rev}(F)} = \underbrace{G(1/x)x^n}_{\mathrm{rev}(G)} \cdot \underbrace{Q(1/x)x^{N-n}}_{\mathrm{rev}(Q)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{N-\deg(R)}$$

Algorithm:

- Compute $\mathrm{rev}(Q) = \mathrm{rev}(F)/\mathrm{rev}(G) \mod x^{N-n+1}$ $O(\mathsf{M}(N))$
- Recover Q $O(N)$
- Deduce $R = F - QG$ $O(\mathsf{M}(N))$

Application: extension of recurrences

[Shoup, 1991]

Problem: Given $N \in \mathbb{N}$ and the first n terms $u_0, \ldots, u_{n-1}$ of a recurrent sequence with constant coefficients of order n, compute $u_n, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(N^2)$

Idea: $\sum_{i \geq 0} u_i x^i$ is rational $A(x)/B(x)$, with B given by the input recurrence, and $\deg(A) < \deg(B)$

Example (Fibonacci): $F_{i+2} = F_{i+1} + F_i \quad \Longleftrightarrow \quad \sum_i F_i x^i = \frac{F_0 + (F_1 - F_0)x}{1 - x - x^2}$

Algorithm:

- Compute A from B and $u_0, \ldots, u_{n-1}$ $O(\mathsf{M}(n))$
- Expand A/B modulo x^{N+1} $O(\mathsf{M}(N))$

Exponentials of power series and 1st order LDE

[Brent, 1975]

To compute the exponential $\exp(f) = \sum_{i\geq 0} \frac{f^i}{i!}$, choose $\varphi(g) = \log(g) - f$:

$$g_0 = 1 \quad \text{and} \quad g_{\kappa+1} = g_\kappa - g_\kappa \left(\log(g_\kappa) - f\right) \mod x^{2^{\kappa+1}} \quad \text{for } \kappa \geq 0.$$

Master Theorem: $\mathsf{C}(N) = \mathsf{C}(N/2) + O(\mathsf{M}(N)) \implies \mathsf{C}(N) = O(\mathsf{M}(N))$

Corollary: Solve first order linear differential equations $af' + bf = c$ in $O(\mathsf{M}(N))$

- if $c = 0$ then the solution is $f_0 = \exp\left(-\int b/a\right)$ $\quad O(\mathsf{M}(N))$
- else, variation of constants: $f = f_0 g$, where $g' = c/(af_0)$ $\quad O(\mathsf{M}(N))$

▶ main difficulty for higher orders: for non-commutativity reasons, the matrix exponential $Y(x) = \exp(\int A(x))$ is not a solution of $Y' = A(x)Y$.

Application: conversion coefficients $\leftrightarrow$ power sums

[Schönhage, 1982]

Any polynomial $F = x^n + a_1 x^{n-1} + \cdots + a_n$ in $\mathbb{K}[x]$ can be represented by its first n power sums $S_i = \sum_{F(\alpha)=0} \alpha^i$

Conversions coefficients $\leftrightarrow$ power sums can be performed

- either in $O(n^2)$ using Newton identities (naive way):

$$i a_i + S_1 a_{i-1} + \cdots + S_i = 0, \quad 1 \leq i \leq n$$

- or in $O(\mathsf{M}(n))$ using generating series

$$\frac{\mathsf{rev}(F)'}{\mathsf{rev}(F)} = -\sum_{i\geq 0} S_{i+1} x^i \quad \Longleftrightarrow \quad \mathsf{rev}(F) = \exp\left(-\sum_{i\geq 1} \frac{S_i}{i} x^i\right)$$

Application: special bivariate resultants

[B-Flajolet-Salvy-Schost, 2006]

Composed products and sums: manipulation of algebraic numbers

$$F \otimes G = \prod_{F(\alpha)=0, G(\beta)=0} (x - \alpha\beta), \quad F \oplus G = \prod_{F(\alpha)=0, G(\beta)=0} (x - (\alpha+\beta))$$

Output size: $N = \deg(F)\deg(G)$

Linear algebra: χ_{xy}, χ_{x+y} in $\mathbb{K}[x,y]/(F(x), G(y))$ $O(\mathsf{MM}(N))$

Resultants: $\mathrm{Res}_y\left(F(y), y^{\deg(G)} G(x/y)\right), \ \mathrm{Res}_y\left(F(y), G(x-y)\right)$ $O(N^{1.5})$

Better: $\otimes$ and $\oplus$ are easy in Newton representation $O(\mathsf{M}(N))$

$$\sum \alpha^s \sum \beta^s = \sum (\alpha\beta)^s \qquad \text{and}$$

$$\sum \frac{\sum(\alpha+\beta)^s}{s!} x^s = \left(\sum \frac{\sum \alpha^s}{s!} x^s\right)\left(\sum \frac{\sum \beta^s}{s!} x^s\right)$$

Corollary: Fast polynomial shift $P(x+a) = P(x) \oplus (x+a)$ $O(\mathsf{M}(\deg(P)))$

Newton for differential systems – a glimpse

The previous iteration for $\exp(f)$ rewrites

$$g_0 = 1 \quad \text{and} \quad g_{\kappa+1} = g_\kappa - g_\kappa \int g_\kappa^{-1} \left(g_\kappa' - f' g_\kappa\right) \mod x^{2^{\kappa+1}}.$$

➥ It computes simultaneously $\exp(f)$ and $\exp(-f)$.

Idea [BCOSSS, 2007]: the similar iteration on polynomial matrices

$$Y_0 = I_r \quad \text{and} \quad Y_{\kappa+1} = Y_\kappa - Y_\kappa \int Y_\kappa^{-1} \left(Y_\kappa' - A Y_\kappa\right) \mod x^{2^{\kappa+1}}.$$

computes (truncated) solutions of $Y' = A(x)Y$ and of $Z' = -ZA(x)$.

Newton iteration on power series matrices

To solve an equation $\phi(Y) = 0$, with $\phi : \mathcal{M}_r(\mathbb{K}[[x]]) \to \mathcal{M}_r(\mathbb{K}[[x]])$:

Define the sequence $Y_{\kappa+1} = Y_\kappa - U_{\kappa+1}$, where

- $U_{\kappa+1}$ is a solution of valuation $\geq 2^{\kappa+1}$ of the linearized equation
$$D\phi|_{Y_\kappa} \cdot U = \phi(Y_\kappa),$$
- $D\phi|_{Y_\kappa}$ is the differential of ϕ at Y_κ.

Then, the sequence Y_κ converges quadratically to the solution Y.

Examples:

- inversion of power series matrices $A(x)^{-1}$
- quasi-exponential of power series matrices $Y' = A(x)Y$

Application: inversion of power series matrices

[Schulz, 1933]

To compute the inverse Z of a matrix of power series $Y \in \mathcal{M}_r(\mathbb{K}[[x]])$:

- choose the map $\phi : Z \mapsto I - YZ$ with differential $Z \mapsto -YZ$
- the equation for U becomes $\quad -YU = I - YZ_\kappa \mod x^{2^{\kappa+1}}$
- solution $\quad U = -Y^{-1}\,(I - YZ_\kappa) = -Z_\kappa\,(I - YZ_\kappa) \bmod x^{2^{\kappa+1}}$

This yields the following Newton-type iteration for Y^{-1}

$$Z_{\kappa+1} = Z_\kappa + Z_\kappa(I_r - YZ_\kappa) \mod x^{2^{\kappa+1}}$$

Master Theorem:

$$\mathsf{C}_{\mathrm{inv}}(N) = \mathsf{C}_{\mathrm{inv}}(N/2) + O(\mathsf{M}(r,N)) \qquad \Longrightarrow \qquad \mathsf{C}_{\mathrm{inv}}(N) = O(\mathsf{M}(r,N))$$

Application: quasi-exponential of power series matrices

[B-Chyzak-Ollivier-Salvy-Schost-Sedoglavic 2007]

To compute the solution $Y \in \mathcal{M}_r(\mathbb{K}[[x]])$ of the system $Y' = AY$

- choose the map $\phi : Y \mapsto Y' - AY$, with differential ϕ.
- the equation for U is $\quad U' - AU = Y'_\kappa - AY_\kappa \mod x^{2^{\kappa+1}}$
- the method of variation of constants yields the solution
  $U = Y_\kappa V_\kappa \bmod x^{2^{\kappa+1}}, \; Y'_\kappa - AY_\kappa = Y_\kappa V'_\kappa \bmod x^{2^{\kappa+1}}$

This yields the following Newton-type iteration for Y:

$$Y_{\kappa+1} = Y_\kappa - Y_\kappa \int Y_\kappa^{-1} \left(Y'_\kappa - AY_\kappa\right) \mod x^{2^{\kappa+1}}$$

Master Theorem:

$\mathsf{C}_{\text{solve}}(N) = \mathsf{C}_{\text{solve}}(N/2) + O(\mathsf{M}(r, N)) \quad \Longrightarrow \quad \mathsf{C}_{\text{solve}}(N) = O(\mathsf{M}(r, N))$

Decrease and conquer III

Recurrences with constant coefficients

Binary powering

Problem: Given a ring $\mathbb{A}$, $a \in \mathbb{A}$ and $N \geq 1$, compute a^N

Naive algorithm: $O(N)$

Better algorithm (Pingala, 200 BC): $O\big(\log N\big)$

Compute a^N recursively, using square-and-multiply

$$a^N = \begin{cases} (a^{N/2})^2, & \text{if } N \text{ is even,} \\ a \cdot (a^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

Modular exponentiation:

- $\mathbb{A} = \mathbb{Z}/A\mathbb{Z}$ N-th decimal of $1/A$ via $(10^N \bmod A)$ in $O\big(\log N\big)$
- $\mathbb{A} = \mathbb{K}[x]/(P)$ if $P, Q \in \mathbb{K}[x]_{<d}$, then $\big(Q^N \bmod P\big)$ in $O\big(\mathsf{M}(d) \log N\big)$

Fibonacci sequence: N-th term

$$F_{n+2} = F_{n+1} + F_n, \quad n \geq 0, \qquad F_0 = 1,\ F_1 = 1.$$

Naive algorithm: N ops. to compute F_N.

Folckore trick:

$$\begin{bmatrix} F_N \\ F_{N+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}}_{C} \begin{bmatrix} F_{N-1} \\ F_N \end{bmatrix} = C^N \begin{bmatrix} F_0 \\ F_1 \end{bmatrix}, \qquad N \geq 1.$$

Binary powering: compute C^N recursively, using

$$C^N = \begin{cases} (C^{N/2})^2, & \text{if } N \text{ is even,} \\ C \cdot (C^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

Cost: $O(\log N)$ products of 2×2 matrices $\longrightarrow$ $O(\log N)$ ops. to compute F_N.

Fibonacci sequence: N-th term

$$\begin{bmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{bmatrix} = C^n \quad\Longrightarrow\quad \begin{bmatrix} F_{2n-2} & F_{2n-1} \\ F_{2n-1} & F_{2n} \end{bmatrix} = \begin{bmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{bmatrix}^2$$

The previous algorithm computes (by squarings of 2×2 symetric matrices)

$$(F_0, F_1, F_2) \to (F_2, F_3, F_4) \to (F_6, F_7, F_8) \to (F_{14}, F_{15}, F_{16}) \to \dots$$

Cost: $5 \times$ and $3 +$ per arrow

Shortt's variant (1978): uses $\begin{cases} F_{2n-2} & = F_{n-2}^2 + F_{n-1}^2 \\ F_{2n-1} & = F_{n-1}^2 + 2F_{n-1}F_{n-2} \end{cases}$ and computes

$$(F_0, F_1) \to (F_2, F_3) \to (F_6, F_7) \to (F_{14}, F_{15}) \to \dots$$

Cost: $3 \times$ and $3 +$ per arrow

Fibonacci sequence: N-th term

Fiduccia's algorithm (1985): binary powering in the ring $\mathbb{K}[x]/(x^2-x-1)$:

$$\begin{aligned} C^n = &\quad \text{matrix of} \quad (x^n \bmod x^2-x-1) \\ \Longrightarrow \quad F_{n-2}+xF_{n-1} = &\quad x^n \bmod x^2-x-1 \end{aligned}$$

Cost: $O(\log N)$ products in $\mathbb{K}[x]/(x^2-x-1) \longrightarrow O(\log N)$ ops. for F_N

Explains Shortt's algorithm:

$$F_{2n-2}+xF_{2n-1} = (F_{n-2}+xF_{n-1})^2 \bmod x^2-x-1$$

N-th term, general case

$$a_{n+d} = c_{d-1}a_{n+d-1} + \cdots + c_0 a_n, \qquad n \geq 0,$$

rewrites

$$\underbrace{\begin{bmatrix} a_N \\ a_{N+1} \\ \vdots \\ a_{N+d-1} \end{bmatrix}}_{v_N} = \underbrace{\begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{d-1} \end{bmatrix}}_{C^T} \underbrace{\begin{bmatrix} a_{N-1} \\ a_N \\ \vdots \\ a_{N+d-2} \end{bmatrix}}_{v_{N-1}} = (C^T)^N \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{bmatrix}}_{v_0}, \qquad N \geq 1.$$

Folklore trick: compute $(C^T)^N$ by binary powering $O(\mathsf{MM}(d)\log(N))$

Fiduccia's algorithm: binary powering in $\mathbb{K}[x]/(P)$, with $P = x^d - \sum_{i=0}^{d-1} c_i x^i$

$$a_N = e \cdot v_N = \left(C^N \cdot e^T\right)^T \cdot v_0 = \langle\, x^N \bmod P,\ v_0 \,\rangle$$

where $e = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$.

Cost: $O(\log N)$ products in $\mathbb{K}[x]/(P)$ $O(\mathsf{M}(d)\log N)$

High order lifting: statement

Problem: Given an invertible polynomial matrix A of degree d, compute the high order components $(C_0, C_1), (C_2, C_3), (C_6, C_7), (C_{14}, C_{15}), \dots$ in the Taylor expansion of its inverse

$$A^{-1} = \sum_{i \geq 0} C_i P^i, \qquad \text{with} \quad P = x^d, \quad C_i \in \mathcal{M}_n\left(\mathbb{K}[x]_{<d}\right)$$

Particular cases:

- If $d = 1$ and $A = I_n - xM$, then $C_i = M^i$, and the high order components can be computed fast by binary powering $O(\mathsf{MM}(n)\,\log(N))$
- If $n = 1$, then high order component $=$ N-th term of a recurrent sequence $\longrightarrow$ can be computed fast by Fiduccia's algorithm $O(\mathsf{M}(d)\,\log(N))$

Upcoming: Storjohann's algorithm $O(\mathsf{MM}(n, d)\,\log(N))$

Generalized Newton identity

Theorem (generalized Newton identity) The following holds modulo P^{s+t+2} :

$$C_{s+1}P^{s+1}+\cdots+C_{s+t+1}P^{s+t+1} = \left(C_0+\cdots+C_tP^t\right)\cdot(I_n - A\cdot(C_0+\cdots+C_sP^s))$$

Particular case: If $s = t = 2^i$, we recover Schulz's Newton-type iteration

Proof:

$$I - A(C_0 + C_1P + \cdots + C_sP^s) = AP^{s+1}(C_{s+1} + C_{s+2}P + \cdots)$$

$$\implies \quad \text{RHS} = \underbrace{(C_0 + C_1P + \cdots + C_tP^t)A}_{I-P^{t+1}(C_{t+1}+\cdots)A}\cdot P^{s+1}(C_{s+1} + C_{s+2}P + \cdots)$$

$$= \text{LHS} \bmod P^{s+t+2}$$

High order lifting: algorithm

[Storjohann, 2002]

Corollary (Storjohann 2002): For all $s, t \geq 0$:

$$C_{s+t+1} = -\{\ (C_{t-1} + C_t P) \cdot \{A \cdot C_s\}\ \}$$

Here, $\{\ B\ \}$ denotes the coefficient of P^1 in B.

Corollary (Storjohann 2002): For all $i \geq 2$, the following equalities hold

$$\begin{cases} C_{2^i-2} & = \ -\ \{\ (C_{2^{i-1}-2} + C_{2^{i-1}-1} P) \cdot \{A \cdot C_{2^{i-1}-2}\}\ \}, \\ C_{2^i-1} & = \ -\ \{\ (C_{2^{i-1}-2} + C_{2^{i-1}-1} P) \cdot \{A \cdot C_{2^{i-1}-1}\}\ \}, \end{cases}$$

and allow to compute the *high order components*

$$(C_0, C_1) \to (C_2, C_3) \to (C_6, C_7) \to (C_{14}, C_{15}) \to \dots$$

Cost: $O(\mathsf{MM}(n,d))$ ops. per arrow $O(\mathsf{MM}(n,d)\ \log(N))$

Generalizes simultaneously binary powering ($d = 1$) and Fiduccia ($n = 1$)

Example (Fibonacci revisited)

$$\frac{1}{1-x-x^2} = C_0 + C_1 P + C_2 P^2 + \cdots, \qquad P = x^2, \quad C_n = F_{2n} + F_{2n+1} x$$

The Storjohann identities become

$$\begin{cases} C_{2^i-2} & = & -\left\{ \left(C_{2^{i-1}-2} + C_{2^{i-1}-1} x^2\right) \cdot \left\{ \left(1-x-x^2\right) \cdot C_{2^{i-1}-2} \right\} \right\}, \\ C_{2^i-1} & = & -\left\{ \left(C_{2^{i-1}-2} + C_{2^{i-1}-1} x^2\right) \cdot \left\{ \left(1-x-x^2\right) \cdot C_{2^{i-1}-1} \right\} \right\}, \end{cases}$$

and allow to compute the *high order components*

$$(F_0, F_1, F_2, F_3) \to (F_4, F_5, F_6, F_7) \to (F_{12}, F_{13}, F_{14}, F_{15}) \to \dots \quad \text{by}$$

$$\begin{cases} F_{2^{i+1}-2} = F_{2^i-2} \cdot F_{2^i} + F_{2^i-1} \cdot F_{2^i-3}, \\ F_{2^{i+1}-4} = F_{2^i-2}^2 + F_{2^i-3}^2, \\ F_{2^{i+1}-1} = F_{2^i-1} \cdot F_{2^i} + F_{2^i-2} \cdot F_{2^i-1}, \\ F_{2^{i+1}-3} = F_{2^i-1} \cdot F_{2^i-2} + F_{2^i-2} \cdot F_{2^i-3}, \end{cases} \iff \underbrace{\begin{cases} F_{2n-2} = F_{n-2}^2 + F_{n-1}^2 \\ F_{2n-1} = F_{n-1}^2 + 2F_{n-1}F_{n-2} \end{cases}}_{\text{Shortt's algorithm}}$$

Keller-Gehrig algorithm

Problem: Given $M \in \mathcal{M}_n(\mathbb{K})$ and $v \in \mathbb{K}^n$, compute the Krylov sequence

$$\mathcal{K} = \Big(v, \quad Mv, \quad M^2 v, \quad \dots, \quad M^{n-1}v\Big)$$

Interest: If M is *generic*, $\mathcal{K}$ forms a basis of $\mathbb{K}^n$, and the matrix C of $v \mapsto Mv$ w.r.t. $\mathcal{K}$ is companion $\Longrightarrow$ the characteristic polynomial $\det(xI_n - M)$ reads off $C = P^{-1}MP$, where $P = \big[\, v \mid Mv \mid \cdots \mid M^{n-1}v \,\big]$.

Naive algorithm: Compute iteratively $v_{i+1} = Mv_i$, $v_0 = v$.

Cost: $O(n)$ matrix-vector products $\longrightarrow O(n^3)$ ops. in $\mathbb{K}$.

Keller-Gehrig algorithm (1985) Compute

(1) $M_0 = M,\ M_1 = M^2,\ M_2 = M^4,\ M_3 = M^8, \dots$ (binary powering)

(2) $\Big[\, M^{2^k}v \mid \cdots \mid M^{2^{k+1}-1}v \,\Big] := M_k \times \Big[\, v \mid Mv \mid \cdots \mid M^{2^k-1}v \,\Big], \quad k \geq 0$

Cost: $O(\log(n))$ matrix products for both (1) and (2) $\longrightarrow O(\mathsf{MM}(n)\log n)$

Solving linear systems with polynomial coefficients

Problem: Given $A \in \mathcal{M}_n(\mathbb{K}[x]_{\leq d})$ invertible, and $b \in \mathbb{K}[x]^n_{<d}$, compute $A^{-1}b$

[Moenck-Carter, 1979]: One can recover the exact solution $y = A^{-1}b \in \mathbb{K}(x)^n$ of $Ay = b$ from its approximation

$$y_{2nd} = A^{-1}b \bmod x^{2nd} = c_0 + c_1 P + c_2 P^2 + \cdots + c_{2n} P^{2n}, \qquad \text{where } \; c_i \in \mathbb{K}[x]^n_{<d}$$

Conversion $y_{2nd} \to y$ by Padé approximation $\qquad O(n\,\mathsf{M}(nd)\log(nd))$

Particular case: If $d = 1$ and $A = I_n - xM$, with $M \in \mathcal{M}_n(\mathbb{K})$ and $b \in \mathbb{K}^n$, then $c_i = M^i b$, and $c_0, \ldots, c_{2n}$ can be computed fast by the Keller-Gehrig algorithm.

Storjohann's algorithm

Problem: Given $A \in \mathcal{M}_n(\mathbb{K}[x]_{\leq d})$ invertible, and $b \in \mathbb{K}[x]^n_{<d}$, compute $y_N = A^{-1} b \bmod x^N$

Theorem (Storjohann 2002): For all $s, t \geq 0$:

$$c_{s+t+1} = -\left\{ \, (C_{t-1} + C_t P) \cdot \{ A \cdot c_s \} \, \right\}$$

Here, $\{ \, v \, \}$ denotes the coefficient of P^1 in $v \in \mathbb{K}[x]^n$.

Corollary (Storjohann 2002): For all $i \geq 2$, the following equality holds

$$\left[\, c_{2^k} \mid \cdots \mid c_{2^{k+1}-1} \, \right] = -\left\{ \left(C_{2^k-2} + C_{2^k-1} P \right) \cdot \left\{ A \cdot \left[\, c_0 \mid \cdots \mid c_{2^k-1} \, \right] \right\} \right\},$$

which allows to compute

$$(c_0, c_1) \to (c_2, c_3) \to (c_4, c_5, c_6, c_7) \to (c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}) \to \dots$$

Generalizes Keller-Gehrig's algorithm

Cost: $O(\mathsf{MM}(n,d))$ ops. per arrow $\qquad O(\mathsf{MM}(n,d)\log(n))$

Baby steps / giant steps

Baby steps / giant steps for polynomial evaluation

Problem: Given a $\mathbb{K}$-algebra $\mathbb{A}$, $a \in \mathbb{A}$ and $P \in \mathbb{K}[x]_{<N}$, compute $P(a)$

Horner rule: $O(N)$ products in $\mathbb{A}$

Better algorithm (Paterson-Stockmeyer, 1973): $O(\sqrt{N})$ products in $\mathbb{A}$

Write $P(x) = P_0(x) + \cdots + P_{\ell-1}(x) \cdot (x^\ell)^{\ell-1}$, with $\ell = \sqrt{N}$ and $\deg(P_i) < \ell$

(BS) Compute $a^2, \quad \dots, \quad a^\ell =: b$ $O(\sqrt{N})$ products in $\mathbb{A}$

(GS) Compute $b_0 = 1,\ b_1 = b, \quad \dots, \quad b_{\ell-1} = b^{\ell-1}$ $O(\sqrt{N})$ products in $\mathbb{A}$

Evaluate $c_0 = P_0(a), \quad \dots, \quad c_{\ell-1} = P_{\ell-1}(a)$ no product in $\mathbb{A}$

Return $P(a) = b_0 c_0 + \cdots + b_{\ell-1} c_{\ell-1}$ $O(\sqrt{N})$ products in $\mathbb{A}$

Application: evaluation of $P \in \mathbb{K}[x]_{<N}$ at a matrix in $\mathcal{M}_r(\mathbb{K})$ $O(\sqrt{N}\,\mathsf{MM}(r))$

Baby steps / giant steps, application to factorials

Problem: Compute $N! = 1 \times 2 \times \cdots \times N$

Naive algorithm: unroll the recurrence $O(N)$

Better algorithm (Strassen, 1976): BS-GS strategy $O\big(\mathsf{M}(\sqrt{N}\,)\log N\big)$

(BS) Compute $P = (x+1)(x+2)\cdots(x+\sqrt{N})$ $O\big(\mathsf{M}(\sqrt{N}\,)\log N\big)$

(GS) Evaluate P at $0, \sqrt{N}, 2\sqrt{N}, \ldots, (\sqrt{N}-1)\sqrt{N}$ $O\big(\mathsf{M}(\sqrt{N}\,)\log N\big)$

Return $u_N = P((\sqrt{N}-1)\sqrt{N})\cdots P(\sqrt{N})\cdot P(0)$ $O(\sqrt{N})$

Baby steps / giant steps, application to recurrences

Problem: Compute the N-th term u_N of a P-recursive sequence

$$p_r(n)u_{n+r} + \cdots + p_0(n)u_n = 0, \qquad (n \in \mathbb{N})$$

Naive algorithm: unroll the recurrence $O(N)$

Better algorithm: $U_n = (u_n, \ldots, u_{n+r-1})^T$ satisfies the 1st order recurrence

$$U_{n+1} = \frac{1}{p_r(n)} A(n) U_n \quad \text{with} \quad A(n) = \begin{bmatrix} & p_r(n) & & \\ & & \ddots & \\ & & & p_r(n) \\ -p_0(n) & -p_1(n) & \ldots & -p_{r-1}(n) \end{bmatrix}.$$

$\Longrightarrow u_N$ reads off the matrix factorial $A(N-1)\cdots A(0)$

Chudnovsky-Chudnovsky, 1987: (BS)-(GS) strategy $O\big(\mathsf{M}(\sqrt{N}\,)\log N\big)$

Baby steps / giant steps, application to point counting

Problem: count the number n of solutions of the equation $y^2 = f(x)$ over $\mathbb{F}_p$

Basic idea (Deuring, 1941): if $\deg(f) = 3$, then $(n \bmod p)$ is $-[x^{p-1}]f(x)^{(p-1)/2}$

Explanation: z is a non-zero square in $\mathbb{F}_p$ exactly when $z^{(p-1)/2} = 1$

Generalization (Cartier-Manin, 1956): for a genus-g curve $y^2 = f(x)$, $(n \bmod p)$ reads off the Hasse-Witt matrix $(h_{i,j})_{i,j=1}^{g}$, with $h_{i,j} = [x^{ip-j}]f(x)^{(p-1)/2}$

Corollary (B-Gaudry-Schost, 2007): hyperelliptic point counting / $\mathbb{F}_p$ in $\tilde{O}(\sqrt{p})$

Based on (Flajolet-Salvy, 1997): $h = f^N$ satisfies the differential equation $fh' - Nf'h = 0$, thus its coefficient sequence is P-recursive.

Tellegen’s transposition principle

Tellegen's transposition principle

Let $\mathbf{M}$ *be a* $\mathsf{m} \times \mathsf{n}$ *matrix, with no zero rows and no zero columns. Any linear algorithm of complexity* L *that computes the matrix-vector product* $\mathbf{M}{\cdot}\mathbf{v}$ *can be transformed into a linear algorithm of complexity*

$$\mathsf{L} - \mathsf{n} + \mathsf{m}$$

that computes the transposed matrix-vector product $\mathbf{M}^{\mathbf{T}}{\cdot}\mathbf{w}$.

- A precise formulation depends on the *model of computation.*

A particular case

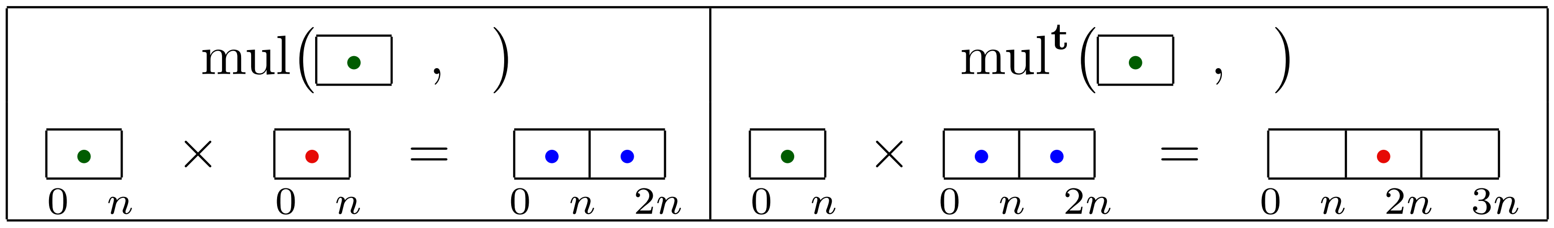
Suppose that the naive algorithm $\mathcal{N}$ is used to compute $M \cdot v$. Define its dual $\mathcal{N}^T$ as the naive algorithm for computing $M^T \cdot w$. Then:

$$\begin{array}{llllllll} \mathcal{N} & \text{uses} & m(n-1) & \text{ops.} & \pm & \text{and} & mn & \text{ops.} \quad \times \\ \mathcal{N}^t & \text{uses} & n(m-1) & \text{ops.} & \pm & \text{and} & mn & \text{ops.} \quad \times. \end{array}$$

$\longrightarrow$ Tellegen's theorem is trivially true for generic matrices

$\longrightarrow$ it becomes interesting when M is structured or sparse

Transposed polynomial multiplication = middle product (MP)



Example:

$$\mathrm{mul}\,(2+3x, a+bx) \qquad \mathrm{mul}^{\mathbf{t}}\,(3+2x, a+bx+cx^2)$$

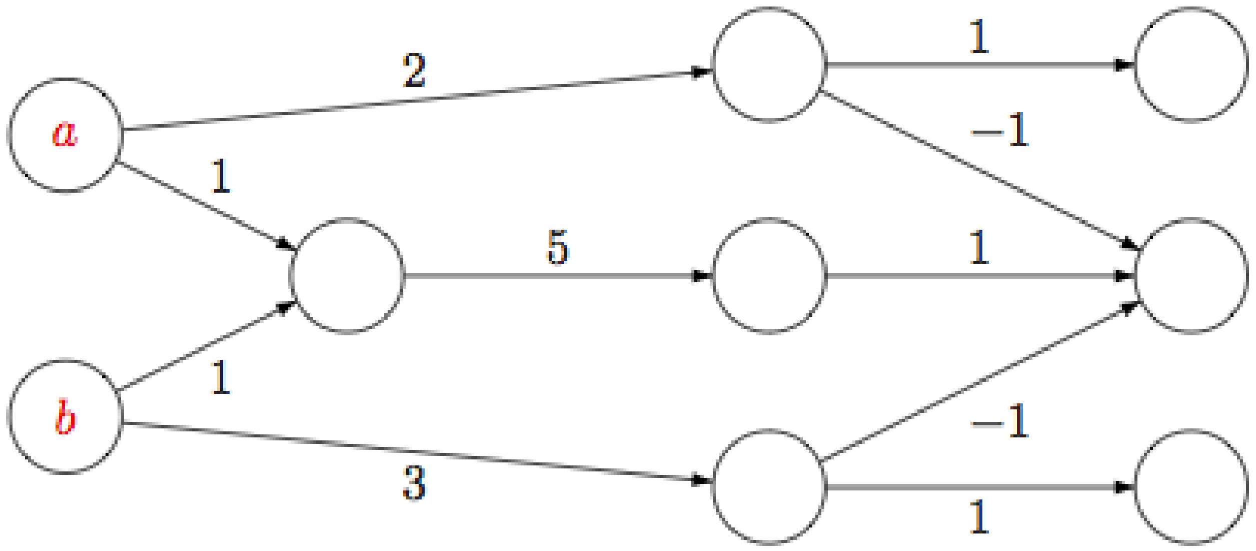
$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix}$$

Theorem [Hanrot-Quercia-Zimmerman, 2002]

From any linear algorithm for multiplication in degree n of cost $\mathsf{M}(\mathbf{n})$, one can derive an algorithm for the $(n, 2n)$ MP, of cost $\mathsf{M}(\mathbf{n}) + \mathbf{O}(\mathbf{n})$.

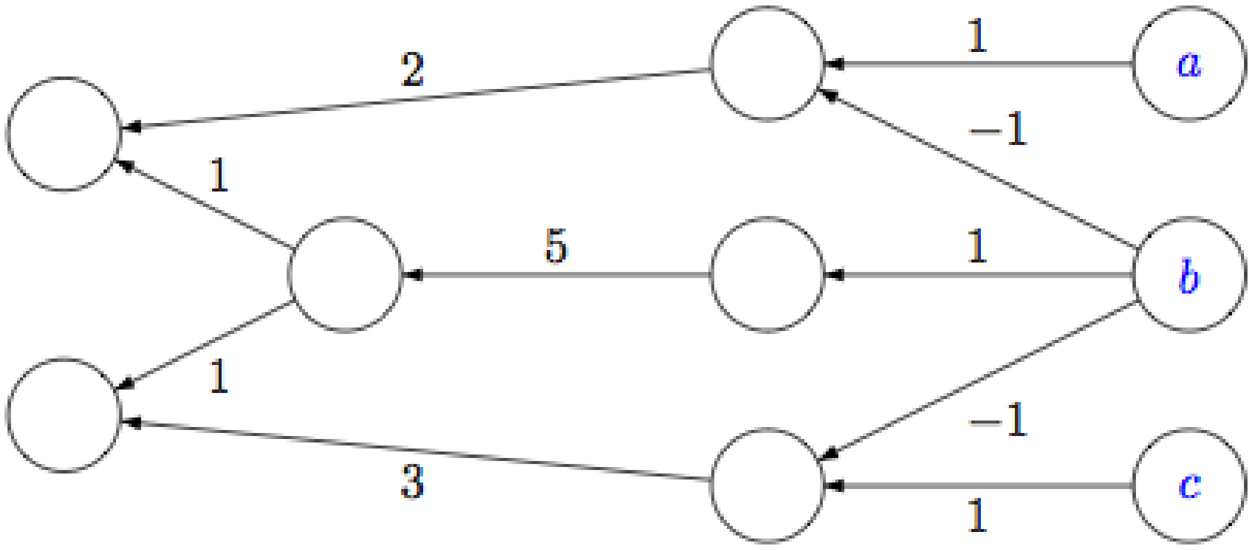
▶ Particular instance of Tellegen's theorem, for Toeplitz-band matrices

A DAG computing $(2+3x)(a+bx)$ *à la Karatsuba*



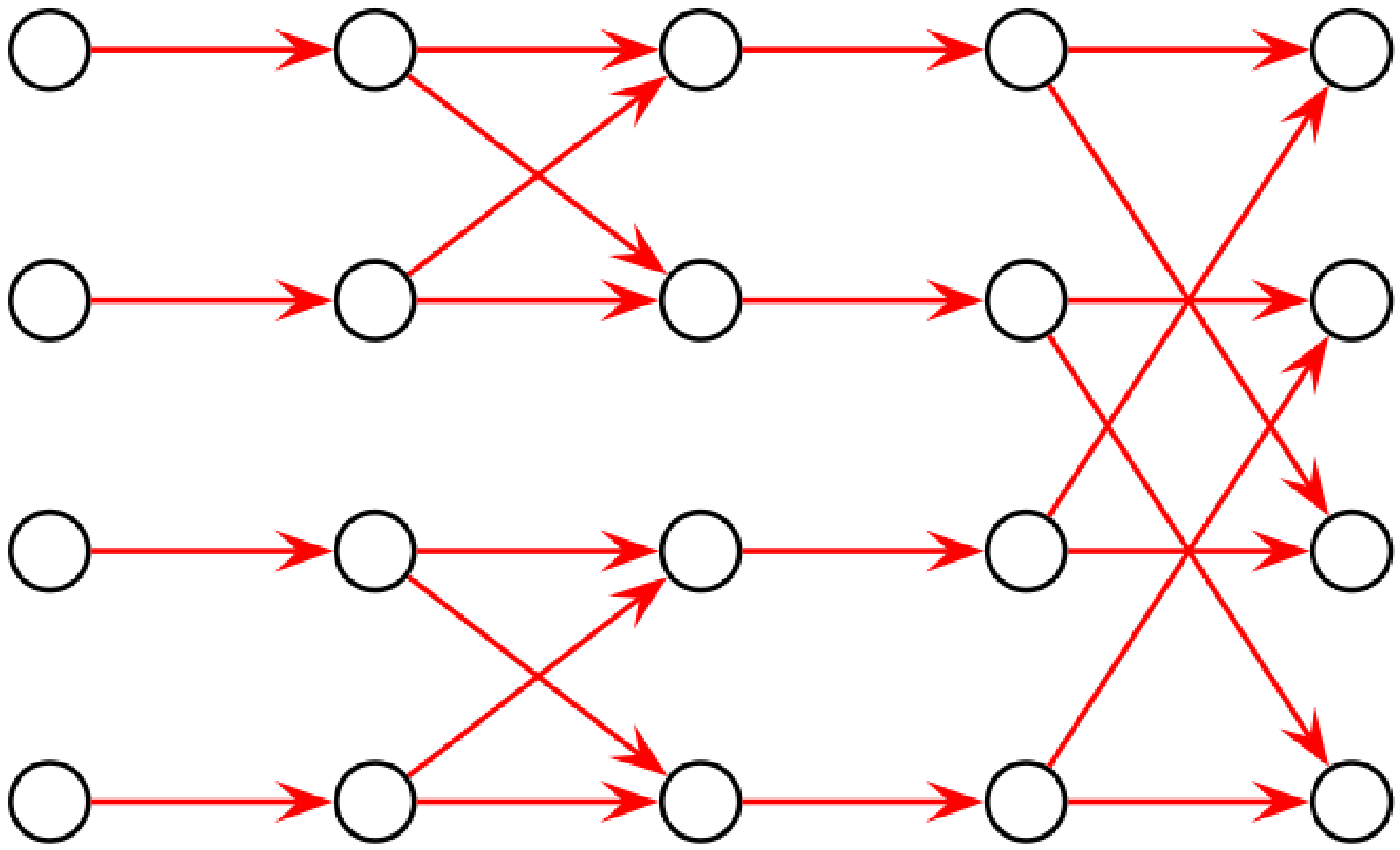
$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix} = \begin{bmatrix} 2a \\ 5(a+b)-2a-3b \\ 3b \end{bmatrix}$$

The transposed DAG computes the middle product of $3 + 2x$ and $a + bx + cx^2$ *à la Karatsuba*



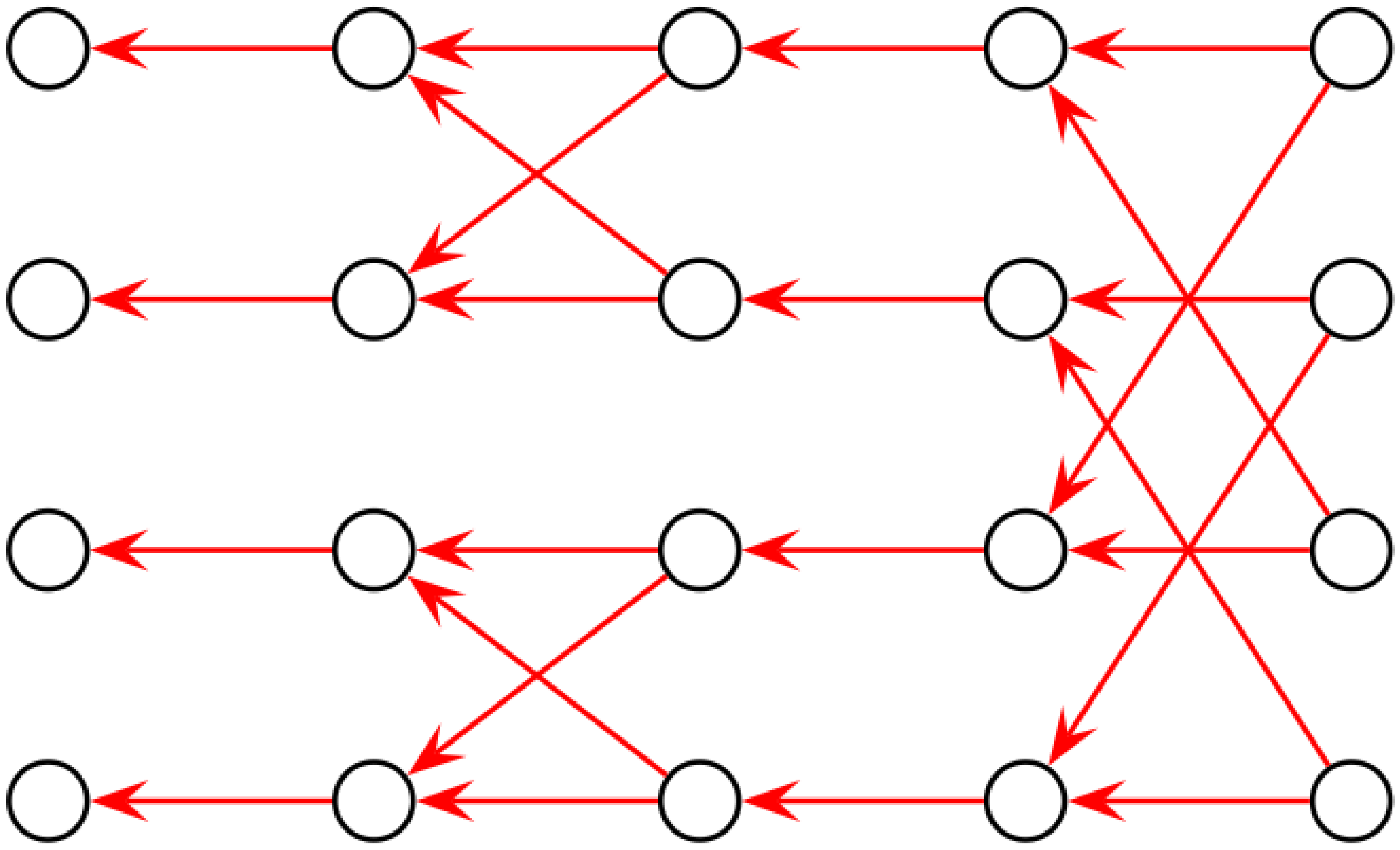
$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a + 3b \\ 2b + 3c \end{bmatrix} = \begin{bmatrix} 2(a-b) + 5b \\ 3(b-c) + 5b \end{bmatrix}$$

Duality between two classes of FFT algorithms



The Cooley-Tukey decimation-in-time DFT, on 4 points

Duality between two classes of FFT algorithms



The Gentleman-Sande decimation-in-frequency DFT, on 4 points

Automatic discovery of Horner's rule

$$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0]$ $\qquad$ $[p_0, \ldots, p_n] \mapsto \sum_{i=0}^n p_i a^i$

```
Input $x_0$.
$p_0 \leftarrow x_0$;
  for $j$ from $1$ to $n$ do
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow ap_j$;
Output $p = [p_0, \ldots, p_n]$.
```

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \textstyle\sum_{i=0}^{n} p_i a^i$$

```
**Input** $x_0$.
$p_0 \leftarrow x_0$;
  **for** $j$ **from** $1$ **to** $n$ **do**
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow a p_j$;
**Output** $p = [p_0, \ldots, p_n]$.
```

```
**Input** $p = [p_0, \ldots p_n]$.




**Output** $x_0$.
```

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0]$

$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$

```
**Input** $x_0$.
$p_0 \leftarrow x_0$;
  **for** $j$ **from** $1$ **to** $n$ **do**
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow a p_j$;
**Output** $p = [p_0, \ldots, p_n]$.
```

```
**Input** $p = [p_0, \ldots p_n]$.
  **for** $j$ **from** $n$ **downto** $1$ **do**




**Output** $x_0$.
```

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^n p_i a^i$$

```
Input $x_0$.
$p_0 \leftarrow x_0$;
  for $j$ from 1 to $n$ do
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow ap_j$;
Output $p = [p_0, \ldots, p_n]$.
```

```
Input $p = [p_0, \ldots p_n]$.
  for $j$ from $n$ downto 1 do
    $p_j \leftarrow ap_j$;
Output $x_0$.
```

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

```
Input $x_0$.
$p_0 \leftarrow x_0$;
  for $j$ from $1$ to $n$ do
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.
```

```
Input $p = [p_0, \ldots p_n]$.
  for $j$ from $n$ downto $1$ do
    $p_j \leftarrow a p_j$;
    $p_{j-1} \leftarrow p_j + p_{j-1}$;

Output $x_0$.
```

Automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

```
**Input** $x_0$.
$p_0 \leftarrow x_0$;
  **for** $j$ **from** $1$ **to** $n$ **do**
    $p_j \leftarrow p_{j-1}$;
    $p_j \leftarrow a p_j$;
**Output** $p = [p_0, \ldots, p_n]$.
```

```
**Input** $p = [p_0, \ldots p_n]$.
  **for** $j$ **from** $n$ **downto** $1$ **do**
    $p_j \leftarrow a p_j$;
    $p_{j-1} \leftarrow p_j + p_{j-1}$;
$x_0 \leftarrow p_0$;
**Output** $x_0$.
```

Karatsuba's algorithm and its transpose

$$\boxed{a \mid b} \times \boxed{c \mid d} \mapsto \underset{|--W--|}{\boxed{U \mid V}} \qquad \boxed{a \mid b} \times^{\mathbf{t}} \underset{|--W--|}{\boxed{U \mid V}} \mapsto \boxed{c \mid d}$$

mul

```
Input $(c, d)$.

$e \leftarrow c + d$ ;
  $U \leftarrow \mathrm{mul}(a, c)$ ;
  $V \leftarrow \mathrm{mul}(b, d)$ ;
  $W \leftarrow \mathrm{mul}(a + b, e)$ ;
$W \leftarrow W - U - V$ ;

Output $(U, V, W)$.
```

Karatsuba's algorithm and its transpose

$$\boxed{\;a\;|\;b\;} \times \boxed{\;c\;|\;d\;} \mapsto \begin{array}{c}\boxed{\quad U \quad|\quad V \quad} \\ |--W--|\end{array} \qquad\qquad \boxed{\;a\;|\;b\;} \times^{\mathbf{t}} \begin{array}{c}\boxed{\quad U \quad|\quad V \quad} \\ |--W--|\end{array} \mapsto \boxed{\;c\;|\;d\;}$$

mul

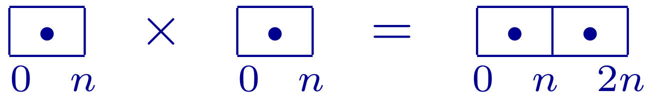
```
**Input** $(c, d)$.

$e \leftarrow c + d$ ;
  $U \leftarrow \mathrm{mul}(a, c)$ ;
  $V \leftarrow \mathrm{mul}(b, d)$ ;
  $W \leftarrow \mathrm{mul}(a + b, e)$ ;
$W \leftarrow W - U - V$ ;

**Output** $(U, V, W)$.
```

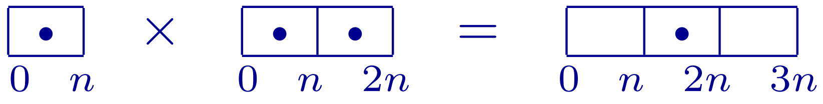
$\mathrm{mul}^{\mathbf{t}}$

```
**Input** $(U, V, W)$.
$V \leftarrow V - W$ ;
$U \leftarrow U - W$ ;
  $e \leftarrow \mathrm{mul}^{\mathbf{t}}(a + b, W)$ ;
  $d \leftarrow \mathrm{mul}^{\mathbf{t}}(b, V)$ ;
  $c \leftarrow \mathrm{mul}^{\mathbf{t}}(a, U)$ ;
$c \leftarrow c + e$ ;
$d \leftarrow d + e$ ;
**Output** $(c, d)$.
```

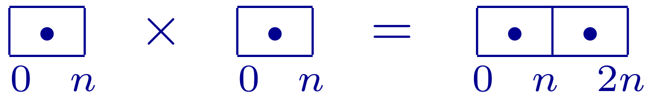
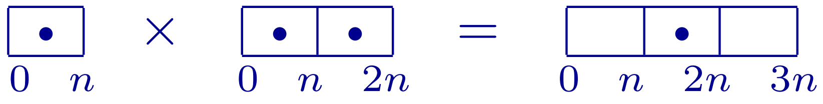
Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product

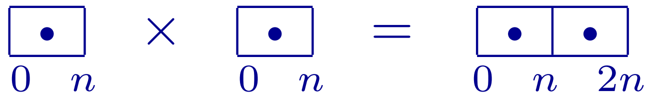
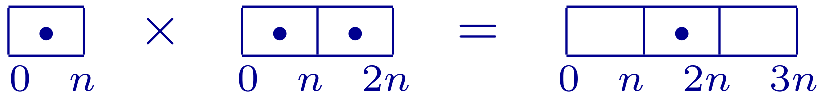




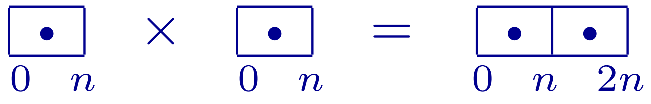
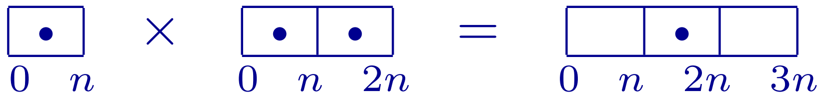
Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \mod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$

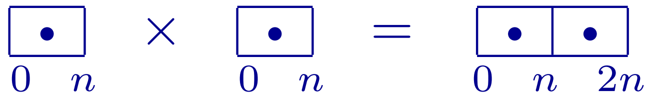
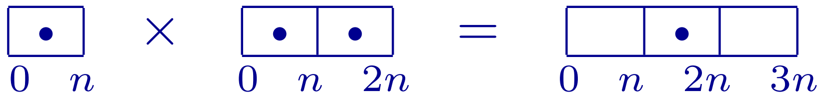
Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \mod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$

Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \mod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$
shift of polynomials	evaluation in falling factorial basis
$P(x) \mapsto P(x+1)$	$P = \sum a_i x^{\underline{i}} \mapsto (P(0), \ldots, P(n-1))$

Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \mod P$	$(a_0, \dots, a_{n-1}) \mapsto (a_0, \dots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \dots, P(a_{n-1}))$	$(p_0, \dots, p_{n-1}) \mapsto (\sum p_i, \dots, \sum p_i a_i^{n-1})$
shift of polynomials	evaluation in falling factorial basis
$P(x) \mapsto P(x+1)$	$P = \sum a_i x^{\underline{i}} \mapsto (P(0), \dots, P(n-1))$
polynomial base change	series composition
$\dots$	$\dots$

Chebyshev polynomials

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

$$\begin{aligned}
T_0(x) &= 1, \\
T_1(x) &= x, \\
T_2(x) &= 2x^2 - 1, \\
T_3(x) &= 4x^3 - 3x, \\
T_4(x) &= 8x^4 - 8x^2 + 1, \\
T_5(x) &= 16x^5 - 20x^3 + 5x, \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1, \\
T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x, \\
T_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1
\end{aligned}$$

Conversion Chebyshev basis $\longrightarrow$ monomial basis

Conversion matrix from the Chebyshev basis to the monomial basis

$$\mathcal{T} = \begin{bmatrix}
1/2 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\
 & 1 & 0 & -3 & 0 & 5 & 0 & -7 & 0 & 9 & 0 \\
 & & 2 & 0 & -8 & 0 & 18 & 0 & -32 & 0 & 50 \\
 & & & 4 & 0 & -20 & 0 & 56 & 0 & -120 & 0 \\
 & & & & 8 & 0 & -48 & 0 & 160 & 0 & -400 \\
 & & & & & 16 & 0 & -112 & 0 & 432 & 0 \\
 & & & & & & 32 & 0 & -256 & 0 & 1120 \\
 & & & & & & & 64 & 0 & -576 & 0 \\
 & & & & & & & & 128 & 0- & 1280 \\
 & & & & & & & & & 256 & 0 \\
 & & & & & & & & & & 512
\end{bmatrix}$$

Question: Perform efficiently the matrix-vector product by $\mathcal{T}$

Transposed Chebyshev conversion

Write:

$$\sum_{n\geq 0} T_n(x)t^n \; = \; \sum_{i\geq 0} C_i(t)x^i.$$

Then, the transposed Chebyshev conversion (in degree $< N$) is the linear map

$$(f_0,\dots,f_{N-1}) \in \mathbb{K}^N \quad \longmapsto \quad f_0C_0(t) + \cdots + f_{N-1}C_{N-1}(t) \bmod t^N$$

General key-fact: (transposed basis conversion amounts to series composition)

$$\sum_{i\geq 0} C_i(t)x^i = \frac{v(t)}{1 - x\cdot h(t)} \quad \Longrightarrow \quad \sum_i C_i(t)f_i \bmod t^N = v(t)\cdot F(h(t)) \bmod t^N$$

Specific key-fact (orthogonality): $\displaystyle \sum_{n\geq 0} T_n(x)t^n = \frac{1}{2}\cdot\frac{1-t^2}{1+t^2}\cdot\frac{1}{1 - x\cdot\frac{2t}{1+t^2}}$

Conclusion: Transposed Chebyshev conversion amounts to composition by $\frac{2t}{1+t^2}$

Fast Chebyshev conversion

Transposed Chebyshev conversion in degree $< N$ means $F\left(\frac{2t}{1+t^2}\right) \bmod t^N$.

Fact: The identity

$$\frac{2t}{1+t^2} = \sqrt{1-\left(\frac{2}{1+t^2}-1\right)^2}$$

implies that the composition $F\left(\frac{2t}{1+t^2}\right) \bmod t^N$ can be performed using a constant number of shifts in degree N.

Consequence: Algorithm of complexity $O(\mathsf{M}(N))$ for transposed Chebyshev conversion in degree $< N$

By Tellegen's principle: Algorithm of complexity $O(\mathsf{M}(N))$ for Chebyshev conversion in degree $< N$

(B-Salvy-Schost 2008): same works for the conversion to a basis $(P_i(x))$ of Sheffer polynomials $\sum_{i\geq 0} P_i(x)\, t^i/i! = v(t)\cdot e^{x\cdot h(t)}$ with "nice" $v(t), h(t)$

Conclusion

Open questions

- Over a general field $\mathbb{K}$, is it possible to compute
  - polynomial products in $\mathbb{K}[x]_{\leq n}$ in $O(n \log n)$? $\quad O(n \log n \log \log n)$
  - matrix products in $\mathcal{M}_r(\mathbb{K})$ in $\tilde{O}(r^2)$? $\quad O(r^{2.38})$
  - compositions in $\mathbb{K}[[x]]/(x^n)$ in $\tilde{O}(n)$? $\quad \tilde{O}(n^{3/2})$
  - resultants in $\mathbb{K}[x,y]_{\leq (n,n)}$ in $\tilde{O}(n^2)$? $\quad \tilde{O}(n^3)$
  - char. polynomials in $\mathcal{M}_r(\mathbb{K}[x]_{\leq d})$ in $\tilde{O}(\mathsf{MM}(r,d))$? $\quad \tilde{O}(r^{2.6972}\, n)$
  - conversions in $\mathbb{K}[x]_{\leq n}$ between bases of special polynomials in $\tilde{O}(n)$?
  - the N-th term of a P-recursive sequence in less than $\tilde{O}(\sqrt{N})$?
  - factorizations in $\mathbb{F}_p[x]_{\leq n}$ in deterministic polynomial time in $(n, \log(p))$?

- Are there $O(\mathsf{M}(n))$ algorithms for
  - evaluation-interpolation at $(1, 2, \ldots, n)$
  - for special gcds, e.g. $\gcd(f, f')$, or $\gcd(f, \prod_{i=1}^{n}(x-k))$
  - rational interpolation at n points in geometric progression

Open questions

- Is DFT optimal? Is polynomial multiplication by FFT optimal?

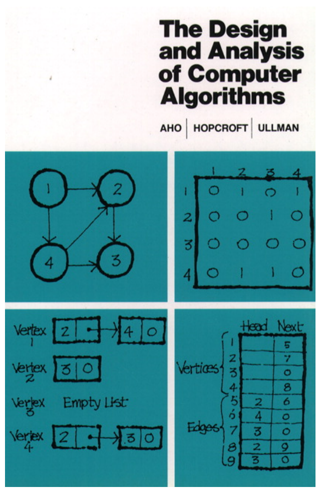
- Can multiplication be done faster for power series than for polynomials? (Is there an FFT version of Mulder's short product?)

- Structured (algebraic / differential) Hermite-Padé approximants

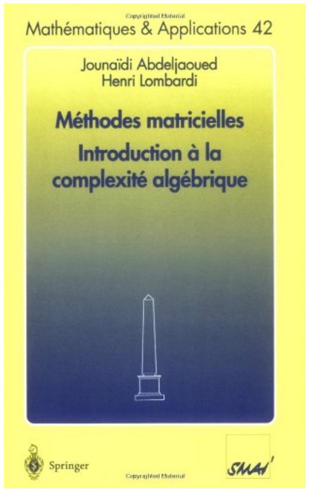
- Evaluation of $P \in \mathbb{K}[x, y]$, or of $L \in \mathbb{K}[x]\langle D_x \rangle$, at $y = y(x) \in \mathbb{K}[[x]]$

- Deterministic $O(\mathsf{MM}(r))$ algorithm for characteristic polynomials in $\mathcal{M}_r(\mathbb{K})$

References


The Design and Analysis of Computer Algorithms

AHO | HOPCROFT | ULLMAN

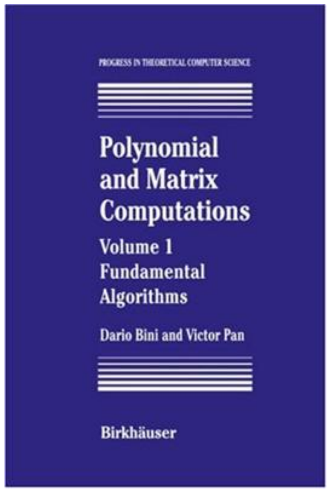

Mathématiques & Applications 42

Jounaïdi Abdeljaoued
Henri Lombardi

Méthodes matricielles
Introduction à la complexité algébrique

Springer

SMAI

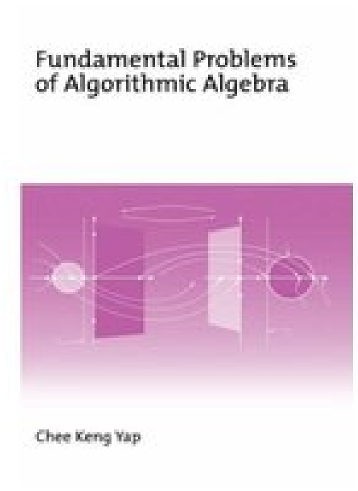

PROGRESS IN THEORETICAL COMPUTER SCIENCE

Polynomial and Matrix Computations

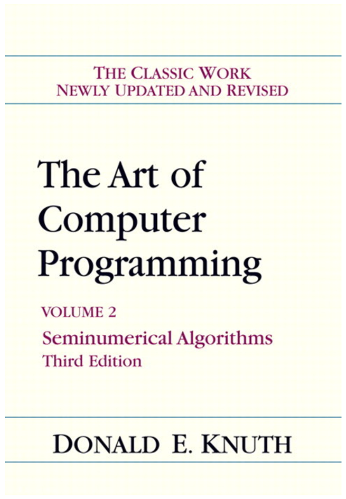
Volume 1
Fundamental Algorithms

Dario Bini and Victor Pan

Birkhäuser


Fundamental Problems of Algorithmic Algebra

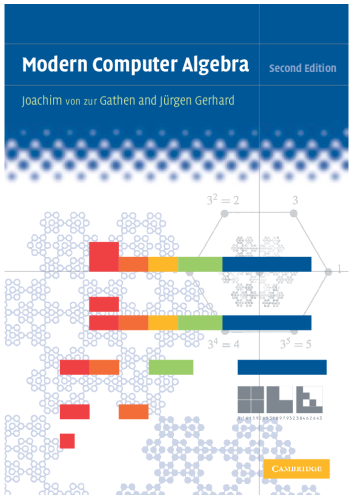
Chee Keng Yap


THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of Computer Programming

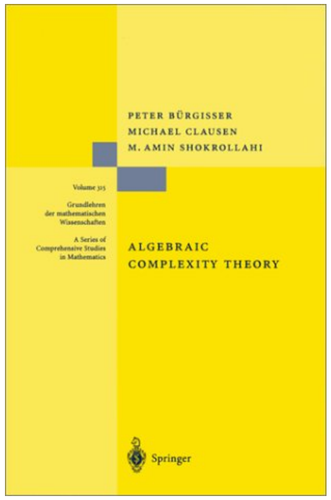
VOLUME 2
Seminumerical Algorithms
Third Edition

DONALD E. KNUTH


Modern Computer Algebra Second Edition

Joachim von zur Gathen and Jürgen Gerhard

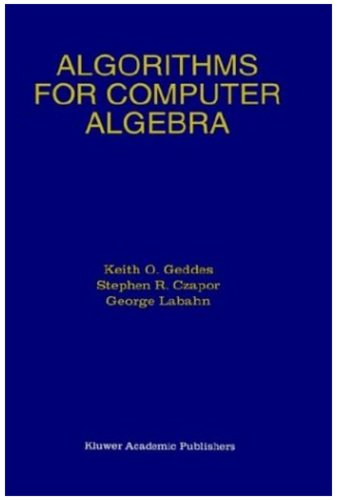
CAMBRIDGE


PETER BÜRGISSER
MICHAEL CLAUSEN
M. AMIN SHOKROLLAHI

Volume 315
Grundlehren der mathematischen Wissenschaften
A Series of Comprehensive Studies in Mathematics

ALGEBRAIC COMPLEXITY THEORY

Springer


ALGORITHMS FOR COMPUTER ALGEBRA

Keith O. Geddes
Stephen R. Czapor
George Labahn

Kluwer Academic Publishers

What can be computed in 1 minute with a CA system

polynomial product[a] in degree 14,000,000 (>1 year with schoolbook)

gcd of two polynomials of degree 600,000

resultant of two polynomials of degree 40,000

factorization of a univariate polynomial of degree 4,000

factorization of a bivariate polynomial of total degree 500

resultant of two bivariate polynomials of total degree 100 (output 10,000)

product/sum of two algebraic numbers of degree 450 (output 200,000)

determinant (char. polynomial) of a matrix with 4,500 (2,000) rows

determinant of an integer matrix with 32-bit entries and 700 rows

product of two integers with 500,000,000 binary digits

factorial of $N = 20,000,000$ (output of 140,000,000 digits);

[a] in $\mathbb{K}[x]$, for $\mathbb{K} = \mathbb{F}_{67108879}$