Algorithms for orthogonal polynomials

A. Bostan	B. Salvy	É. Schost
Algo project	Algo project	ORCCA
INRIA	INRIA	UWO

Overview

Conversion algorithms for univariate polynomials

- Monomial basis
- Lagrange basis
- Newton basis
- Orthogonal bases

Goal: asymptotically fast algorithms for all conversions.

This talk:

- $O(\mathsf{M}(n)\log(n))$ algorithms for monomial vs. orthogonal bases.
- $\mathsf{M}(n)$: cost of multiplying polynomials in degree n.
- We count base field operations.

Previous work

Direct conversion

- Kogge-Stone, Strassen — subproduct-tree techniques
- Driscoll-Healy-Rockmore — one way, $O(\mathsf{M}(n)\log(n))$
- Potts-Steidl-Tasche — one way, floating-point $O(n\log(n)^2)$
- Heinig — more general, $O(\mathsf{M}(n)\log(n)^2)$

Transposition

- Shoup, Kaltofen
- Hanrot-Quercia-Zimmermann — middle product
- with Bostan, Lecerf — Lagrange & Newton interpolation

Orthogonal polynomials

- van Iseghem, Brezinski — vector orthogonal polynomials

  simultaneous Padé

Orthogonal families

Orthogonal polynomials: orthogonal basis with respect to a weight function w

$$\langle p_n, p_m \rangle = \int_a^b w(x) p_n(x) p_m(x) dx.$$

Consequences:

- 3-term recurrence relation $p_{-1} = 0$, $p_0 = 1$ and

$$p_{n+1} = (x - c_{n+1}) p_n - b_{n+1} p_{n-1}.$$

- Writing a polynomial q on the basis (p_i) amounts to compute the scalar products

$$\frac{\langle q, p_i \rangle}{\langle p_i, p_i \rangle}.$$

- Data structure: (b_n) and (c_n) (other possible choices, see later).

Structure of the problem

Let $\mathbf{A}$ be the $n \times n$ matrix of change-of-basis in degree $< n$:

$$\mathbf{A}_{i,j} = \mathsf{coefficient}(p_j, x_i),$$

so that

- the direct conversion is multiplication by $\mathbf{A}$;
- the inverse conversion is multiplication by $\mathbf{A}^{-1}$.

This matrix is structured: the 3-term recurrence implies that the matrix

$$\phi(\mathbf{A}) = \mathbf{A} - (\mathbf{A} \text{ shifted down by one unit}) - (\mathbf{A} \times \text{diagonal, shifted}) - \cdots$$

has small rank.

The standard structured matrices algorithms seem to be unable to deal with this structure.

Expansion

The recurrence on the polynomials p_j is better written in matrix form:

$$\begin{bmatrix} p_j \\ p_{j+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -b_{j+1} & (x - c_{j+1}) \end{bmatrix} \begin{bmatrix} p_{j-1} \\ p_j \end{bmatrix}$$

or

$$\begin{bmatrix} p_j \\ p_{j+1} \end{bmatrix} = \mathbf{M}_{j-1,j} \begin{bmatrix} p_{j-1} \\ p_j \end{bmatrix}$$

More generally:

$$\begin{bmatrix} p_j \\ p_{j+1} \end{bmatrix} = \mathbf{M}_{i,j} \begin{bmatrix} p_i \\ p_{i+1} \end{bmatrix}$$

Expansion: divide and conquer

To compute $a_0p_0 + \cdots + a_7p_7$, we compute

$$\Big([a_0\ a_1]\mathbf{M}_{0,0} + [a_2\ a_3]\mathbf{M}_{0,2} + [a_4\ a_5]\mathbf{M}_{0,4} + [a_6\ a_7]\mathbf{M}_{0,6}\Big) \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$$

and we use

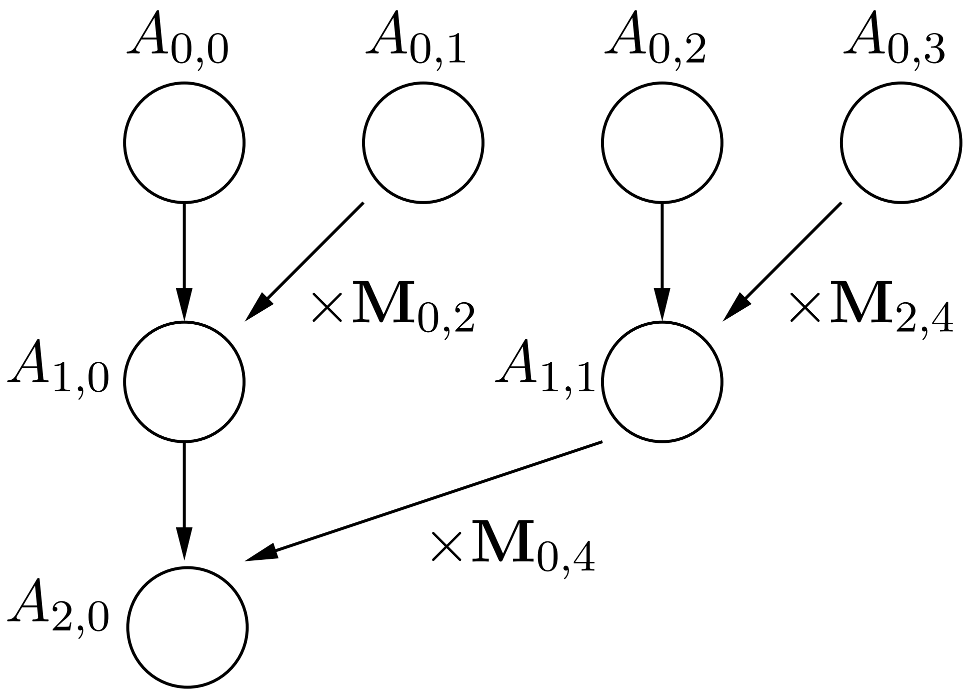
$$[a_0\ a_1]\ \mathbf{M}_{0,0} + [a_2\ a_3]\ \mathbf{M}_{0,2} + [a_4\ a_5]\ \mathbf{M}_{0,4} + [a_6\ a_7]\ \mathbf{M}_{0,6} =$$

$$[a_0\ a_1]\ \mathbf{M}_{0,0} + [a_2\ a_3]\ \mathbf{M}_{0,2} + \Big([a_4\ a_5]\ \mathbf{M}_{4,4} + [a_6\ a_7]\ \mathbf{M}_{4,6}\Big)\mathbf{M}_{0,4}$$

Consequences

- Divide-and-conquer
- Setup a binary tree containing matrices $\mathbf{M}_{i,j}$
- Complexity $O(\mathsf{M}(n)\log(n))$.

Going up the tree



Degrees double as we go towards the root.

Inversion: some nice cases

Some nice families of orthogonal polynomials p_n admit adjoint families p'_n such that

$$\sum_i a_i p_i = \sum_i b_i x^i \iff \sum_i a_i^* x^i = \sum_i b_i p_i^*,$$

where the p_n^* also satisfy a linear recurrence, and a_i^* is “nicely related” to a_i (e.g., $a_i^* = a_i$ or $a_i^* = a_i/i!$).

Example: the Hermite polynomials H_n satisfy

$$H_{n+1} = 2xH_n - 2nH_{n-1};$$

the adjoint family satisfies

$$H_{n+1}^* = 2xH_n^* + 2nH_{n-1}^* \quad \text{with} \quad a_i^* = a_i.$$

Consequence:

- we can reuse the direct conversion algorithm;
- complexity $O(\mathsf{M}(n)\log(n))$.

What are the nice cases?

These are the families of orthogonal polynomials arise as eigenvalues of operators

$$p \mapsto a(x)p'' + b(x)p' \qquad \text{(plus conditions)}.$$

The weight w is such that $(aw)' = bw$; the roots of a give the bounds.

Example: for the Hermite polynomials, $a(x) = 1$ and $b(x) = -x$.

Application. In this case, the polynomials p_n^* are given by

$$p_n^*(u) = \int \frac{w(\alpha)}{1 + ua'(\alpha)} x^n dx$$

with $\alpha - x + ua(\alpha) = 0$.

Using this integral representation, we can then deduce the requested recurrence from a and b; the relationship between a_i^* and a_i is deduced from the norm $\langle p_i, p_i \rangle$.

Inversion: using the orthogonality

Let $\mathbf{A}$ be the $n \times n$ matrix of change-of-basis in degree $< n$:

$$\mathbf{A}_{i,j} = \mathsf{coefficient}(p_j, x_i),$$

so that the direct conversion is multiplication-by-$\mathbf{A}$.

Orthogonality:

$$\mathbf{A}^t \, \mathbf{L} \, \mathbf{A} = \mathsf{diagonal}(e_0, \ldots, e_{n-1}),$$

where

- $e_i = \langle p_i, p_i \rangle = b_1 \cdots b_i$,
- $\mathbf{L}$ is the moment matrix $\mathbf{L}_{i,j} = \langle x^i, x^j \rangle = \langle 1, x^{i+j} \rangle$.

Consequence:

$$\mathbf{A}^{-1} = \mathsf{diagonal}(e_0, \ldots, e_{n-1})^{-1} \; \mathbf{A}^t \; \mathbf{L}^t.$$

Inversion

Step 1. Finding the moment matrix.

- Define $g_j^\star$ as the reciprocal polynomial of p_j.
- Define h_j by

$$h_{j+1} = (1 - c_{j+1}x)h_j - x^2 b_{j+1} h_{j-1}, \quad h_0 = 0, \quad h_1 = 1.$$

Then

$$\frac{h_n}{g_n^\star} = \langle 1, p_0 \rangle + \langle 1, p_1 \rangle x + \langle 1, p_2 \rangle x^2 + \cdots \mod x^{2n}.$$

Conclusion:

- given n coefficients (b_j) and (c_j), one can compute the first $2n$ moments in time $O(\mathsf{M}(n)\log(n))$.

Remark: One can recover the recurrence from the moments using the fast Euclidean algorithm.

Inversion

Step 2. Performing the matrix-vector product.

- Multipication by the matrix $\mathbf{L}^t$ $\mathsf{M}(n)$.

  cf. Hankel matrix.

- Multiplication by the matrix $\mathbf{A}^t$ $O(\mathsf{M}(n)\log(n))$.

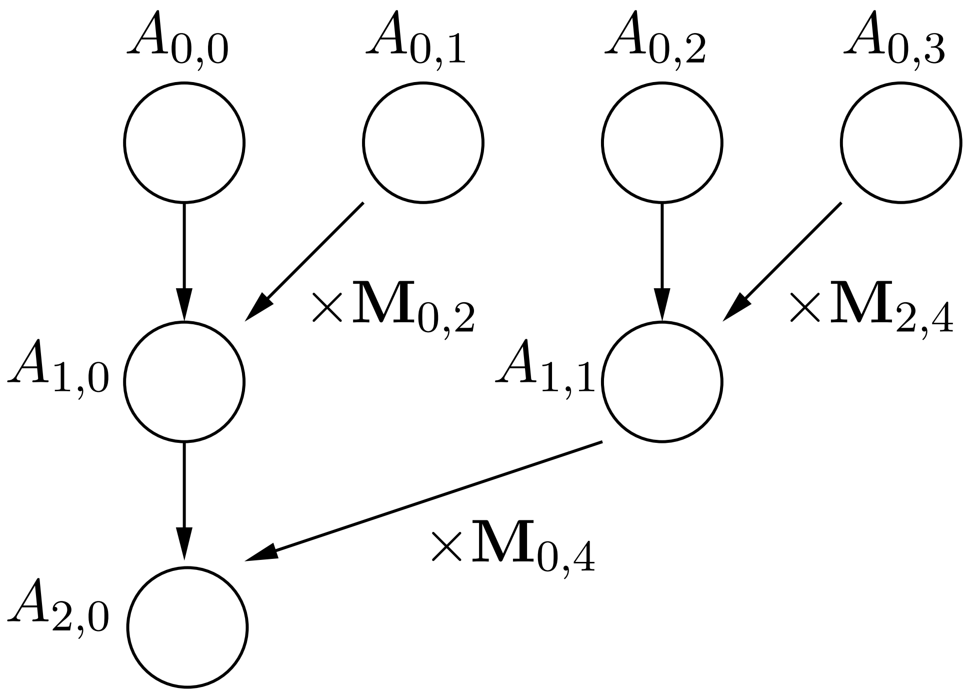
  cf. transposition principle:

  - we have an algorithm of cost $O(\mathsf{M}(n)\log(n))$ for multiplication by $\mathbf{A}$;
  - we deduce an algorithm of cost $O(\mathsf{M}(n)\log(n))$ for multiplication by $\mathbf{A}^t$.

  Very similar to transposed algorithms for Lagrange interpolation (Kaltofen-Lakshman, Bostan-Lecerf-Schost) and Newton interpolation (Bostan-Schost).

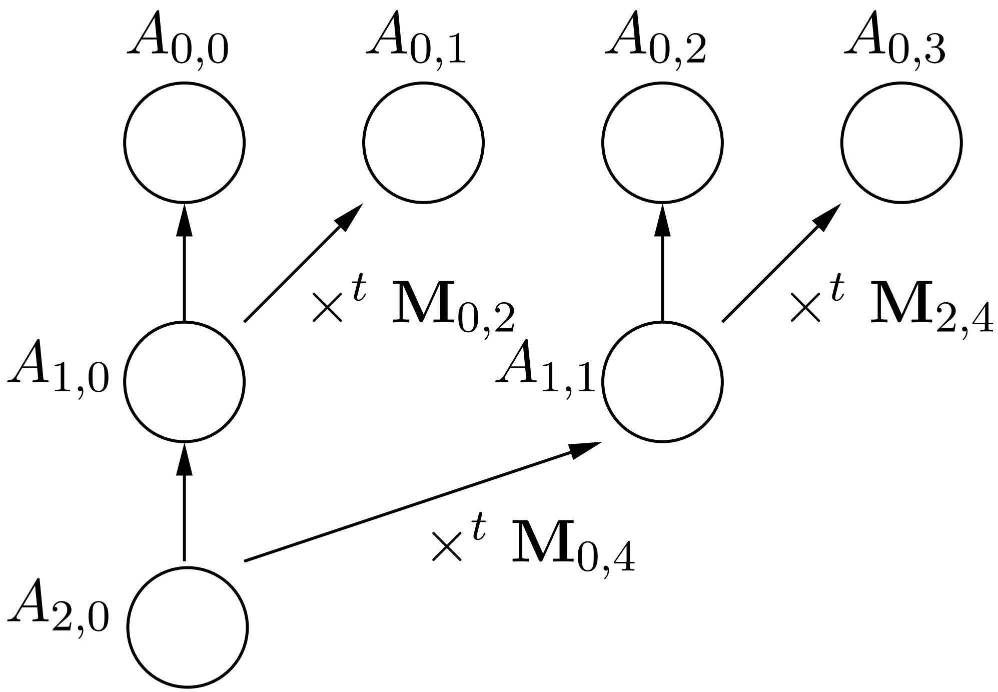
- No interpretation (Bernstein, scaled remainder trees for Lagrange).

Going up the tree



Degrees double as we go towards the root.

Going down the tree

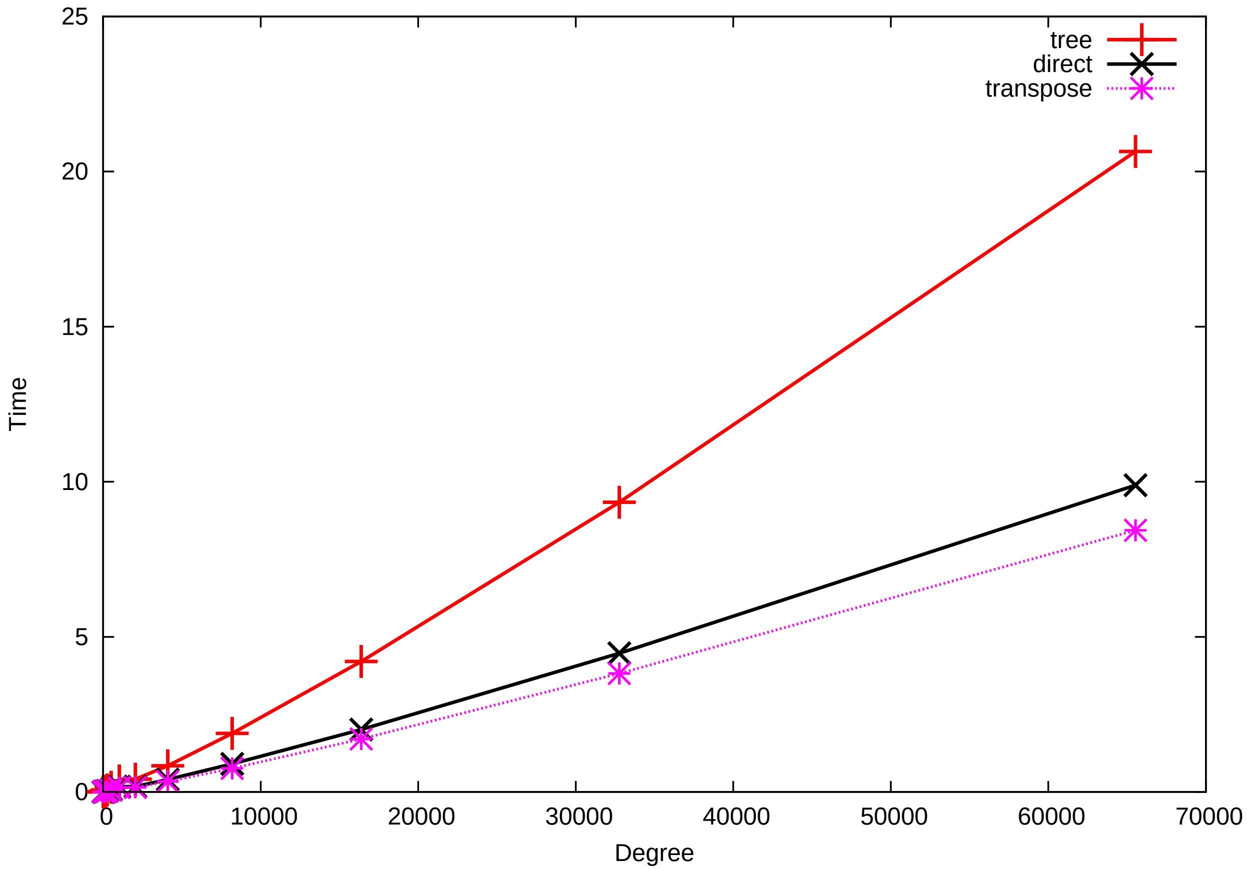


Degrees are halved as we go from the root.

In practice

Experiments:

- Pentium M, 1.7 Ghz
- NTL, prime field with 40 bits (`ZZ_pX`)



Transpose code

Direct

```
  for(long i = depth-2; i >= 0; i--)
    for (long j = 0; j <= length_half-1; j++)
      mul(tmp0, tree[i+1][2*j](0,0), g[2*j+1][0]);
      mul(tmp1, tree[i+1][2*j](1,0), g[2*j+1][1]);
      g[2*j+1][0] += tmp0 + tmp1;             ...
```

Transpose

```
for(long i = 0; i <= depth-2; i++)
    for (long j = length_half-1; j >= 0; j--)
      tmul(tmp0, alpha, tree[i+1][2*j](0,0), arg0);
      tmul(tmp1, alpha, tree[i+1][2*j](0,1), arg1);
      g[2*j+1][0] = tmp0 + tmp1;              ...
```

Going further

Apart from the (still partly mysterious) adjoint polynomials, the algorithms rely only on the 3-term recurrence.

Some special cases behave better.

What about other recurrences, such as

$$p_{n+2} = (x - c_{n+2})p_{n+1} - b_{n+2}p_n - a_{n+2}p_{n-1}?$$

Such recurrences define vector orthogonal polynomials.

The direct conversion extends easily, but not the inverse one: now, we can find two moment matrices with

$$\mathbf{A}^t\,\mathbf{L}_1\,\mathbf{A} = \mathbf{G}_1 \quad \text{and} \quad \mathbf{A}^t\,\mathbf{L}_2\,\mathbf{A} = \mathbf{G}_2,$$

but the matrices $\mathbf{G}_1$ and $\mathbf{G}_2$ are hard to exploit.