Fast Conversion Algorithms for Orthogonal Polynomials

Alin Bostan Bruno Salvy

Algorithms Project, INRIA Paris-Rocquencourt
78153 Le Chesnay Cedex France

and

Éric Schost

ORCCA and Computer Science Department, Middlesex College,
University of Western Ontario, London, Canada

Abstract

We discuss efficient conversion algorithms for orthogonal polynomials. We describe a known conversion algorithm from an arbitrary orthogonal basis to the monomial basis, and deduce a new algorithm of the same complexity for the converse operation.

Key words: Fast algorithms, transposed algorithms, basis conversion, orthogonal polynomials
AMS classification: 2C05, 05E35, 11Y16, 68Q25, 68W40

1 Introduction

Let $(a_i)_{i\geq 1}$, $(b_i)_{i\geq 1}$ and $(c_i)_{i\geq 1}$ be sequences with entries in a field $\mathbb{K}$. We can then define the sequence $(F_i)_{i\geq 0}$ of orthogonal polynomials in $\mathbb{K}[x]$ by $F_{-1} = 0$, $F_0 = 1$ and for $i \geq 1$ by the second order recurrence

$$F_i = (a_i x + b_i)F_{i-1} + c_i F_{i-2}. \tag{1}$$

Email addresses: Alin.Bostan@inria.fr (Alin Bostan), Bruno.Salvy@inria.fr (Bruno Salvy), eschost@uwo.ca (Éric Schost).

Following standard conventions, we require that $a_i c_i$ is non-zero for all $i \geq 1$; in particular, F_i has degree i for all $i \geq 0$ and $(F_i)_{i\geq 0}$ forms a basis of the $\mathbb{K}$-vector space $\mathbb{K}[x]$.

Basic algorithmic questions are then to perform efficiently the base changes between the basis $(F_i)_{i\geq 0}$ and the monomial basis $(x^i)_{i\geq 0}$. More precisely, for $n \in \mathbb{N} \setminus \{0\}$, we study the following problems.

Expansion Problem (Expand_n). Given $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{K}$, compute the coefficients on the monomial basis of the polynomial A defined by the map

$$[\alpha_0, \ldots, \alpha_{n-1}] \mapsto A = \sum_{i=0}^{n-1} \alpha_i F_i \tag{2}$$

Decomposition Problem (Decomp_n). Conversely, given the coefficients of A on the monomial basis, recover the coefficients $\alpha_0, \ldots, \alpha_{n-1}$ in the decomposition (2) of A as a linear combination of the F_i's.

For $i, j \geq 0$, let $F_{i,j}$ be the coefficient of x^i in F_j, and let $\mathbf{F}_n$ be the $n \times n$ matrix with entries $[F_{i,j}]_{0\leq i,j<n}$. Problem Expand_n amounts to multiplying the matrix $\mathbf{F}_n$ by the vector $[\alpha_0, \ldots, \alpha_{n-1}]^t$; hence, the inverse map Decomp_n is well-defined, since $\mathbf{F}_n$ is an upper-triangular matrix whose i-th diagonal entry $F_{0,0} = 1$ (for $i = 0$) and $F_{i,i} = a_1 a_2 \cdots a_i$ (for $i > 0$) is non-zero. As we will see, the *dual problem* (multiplying the matrix $\mathbf{F}_n^t$ by a vector), denoted Expand_n^t, plays an important role as well.

Naive algorithms work in complexity $O(n^2)$ for both problems Expand_n and Decomp_n. Faster algorithms are already known, see details below on prior work. The only new result in this article is the second part of Theorem 1 below; it concerns fast computation of the map Decomp_n.

As usual, we denote by M a *multiplication time* function, such that polynomials of degree less than n in $\mathbb{K}[x]$ can be multiplied in $\mathsf{M}(n)$ operations in $\mathbb{K}$, when written in the monomial basis. Besides, we impose the usual super-linearity conditions of [11, Chap. 8]. Using Fast Fourier Transform algorithms, $\mathsf{M}(n)$ can be taken in $O(n \log(n))$ over fields with suitable roots of unity, and in $O(n \log(n) \log\log(n))$ over any field [20,4].

Theorem 1 *Problems* Expand_n *and* Decomp_n *can be solved in* $O(\mathsf{M}(n) \log(n))$ *arithmetic operations in* $\mathbb{K}$.

The asymptotic estimates of Theorem 1 also hold for conversions between any arbitrary orthogonal bases, using the monomial basis in an intermediate step. In conjunction with FFT algorithms for polynomial multiplication, Theorem 1 shows that all such base changes can be performed in *nearly linear time.*

Previous work. Fast algorithms are known for problems closely related to Problem Expand_n. From these, one could readily infer fast algorithms for Problem Expand_n itself.

In [19], the question is the computation of the values

$$[\alpha_0, \ldots, \alpha_{n-1}] \mapsto \left[\sum_{i=0}^{n-1} \alpha_i F_i(x_j) \right]_{0 \le j < n},$$

where the x_j are the Chebyshev points $x_j = \cos(j\pi/(n-1))$. This is done by expanding $\sum_{i=0}^{n-1} \alpha_i F_i$ on the Chebyshev basis and applying a discrete cosine transform. The article [7] studies the transposed problem: computing the map

$$[\alpha_0, \ldots, \alpha_{n-1}] \mapsto \left[\sum_{i=0}^{n-1} \alpha_i F_j(x_i) \right]_{0 \le j < n}. \tag{3}$$

The algorithm in [7] is (roughly, see [19] for details) the transpose of the one in [19]: it applies a transposed multipoint evaluation, then a transposed conversion, to either the monomial or the Chebyshev basis.

Regarding Problem Decomp_n, to the best of our knowledge, no $O(\mathsf{M}(n)\log(n))$ algorithm has appeared before, except for particular families of polynomials, like Legendre [10], Chebyshev [17] and Hermite [16]. In the case of arbitrary orthogonal polynomials, the best complexity result we are aware of is due to Heinig [14], who gives a $O(\mathsf{M}(n)\log^2(n))$ algorithm for solving inhomogeneous linear systems with matrix $\mathbf{F}_n^t \mathbf{F}_n$. From this, it is possible to deduce an algorithm of the same cost for Problem Decomp_n.

In [19], one sees mentions of left and right inverses for the related problem

$$[\alpha_0, \ldots, \alpha_{n-1}] \mapsto \left[\sum_{i=0}^{n-1} \alpha_i F_i(x_j) \right]_{0 \le j < 2n-1}.$$

In [16], the inverse of the map (3) is discussed: when (x_i) are the roots of F_n, Gauss' quadrature formula shows that this map is orthogonal, so that inversion reduces to transposition. In other cases, approximate solutions are given.

The various algorithms mentioned up to now have costs $O(\mathsf{M}(n)\log(n))$ or $O(\mathsf{M}(n)\log^2(n))$. In [2], we give algorithms of lower cost $O(\mathsf{M}(n))$ for many classical orthogonal polynomials (Jacobi, Hermite, Laguerre, ...), for both Problems Expand_n and Decomp_n.

Main ideas and organization of the paper. Here is a brief description of the strategy used to obtain the complexity estimate of Theorem 1. Three main

ingredients are used: (i) a $O(\mathsf{M}(n)\log(n))$ algorithm for Problem Expand_n; (ii) the transposition principle; (iii) the Favard-Shohat theorem.

The complete treatment with detailed algorithms is given in Sections 2 to 4. In Section 2, we deal with point (i) above: we recast (1) into a first-order matrix recurrence and use standard divide-and-conquer techniques to perform the expansion, as in the algorithm of [12, Th. 2.4] for the conversions between Newton and monomial bases.

An algorithmic theorem called the *transposition principle* [3, Th. 13.20] states that the existence of an algorithm of cost $O(\mathsf{M}(n)\log(n))$ for Expand_n implies the existence of another one *with the same cost* for the dual problem Expand_n^t. In Section 3, we use an effective version of the principle, allowing to design the transposed algorithm in a straightforward manner starting from the direct one.

In Section 4, we solve Problem Decomp_n. Using our algorithm for Problem Expand_n, we first give a constructive version of the Favard-Shohat theorem [8,21]. This enables us to compute the entries of a Hankel matrix $\mathbf{H}_n$ and of a diagonal matrix $\mathbf{D}_n$, reducing Problem Decomp_n to Problem Expand_n^t, up to (inexpensive) pre- and post-multiplication by $\mathbf{H}_n$ and $\mathbf{D}_n^{-1}$.

In the following, we always suppose for simplicity that the number of unknown coefficients n is a power of two.

2 Expansion Problem

We first describe the conversion from the orthogonal basis to the monomial one. As pointed out above, the content of this section is mostly already known. However, our algorithm for the inverse operation rests crucially on this conversion, so we prefer to make it explicit.

Given $\alpha_0, \ldots, \alpha_{n-1}$, we compute here the expansion on the monomial basis of

$$A = \alpha_0 F_0 + \cdots + \alpha_{n-1} F_{n-1}.$$

The ideas are classical; our presentation is taken from [19]. However, our use of "classical" fast multiplication techniques avoids the need of precomputed constants arising in [19], and holds over any field. For $i \geq 0$, define the transition matrix

$$\mathbf{M}^{(i,i+1)} = \begin{bmatrix} 0 & 1 \\ c_{i+1} & a_{i+1}x + b_{i+1} \end{bmatrix},$$

so that we have

$$\begin{bmatrix} F_i \\ F_{i+1} \end{bmatrix} = \mathbf{M}^{(i,i+1)} \begin{bmatrix} F_{i-1} \\ F_i \end{bmatrix}.$$

For $j > i$, let $\mathbf{M}^{(i,j)} = \mathbf{M}^{(j-1,j)}\mathbf{M}^{(j-2,j-1)} \cdots \mathbf{M}^{(i,i+1)}$; for $i = j$, $\mathbf{M}^{(i,j)}$ is the 2×2 identity matrix. It follows that we have

$$\begin{bmatrix} F_{j-1} \\ F_j \end{bmatrix} = \mathbf{M}^{(i,j)} \begin{bmatrix} F_{i-1} \\ F_i \end{bmatrix};$$

besides, for $\ell \geq j \geq i$, we have the associativity relation $\mathbf{M}^{(i,\ell)} = \mathbf{M}^{(j,\ell)}\mathbf{M}^{(i,j)}$. We can then read the polynomial A off the 1×1 matrix

$$\begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} \alpha_0 & \alpha_1 \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix} + \cdots + \begin{bmatrix} \alpha_{n-2} & \alpha_{n-1} \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-1} \end{bmatrix},$$

where the sum has $n/2$ terms. We deduce the equalities

$$\begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} \alpha_0 & \alpha_1 \end{bmatrix} \mathbf{M}^{(1,1)} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix} + \cdots + \begin{bmatrix} \alpha_{n-2} & \alpha_{n-1} \end{bmatrix} \mathbf{M}^{(1,n-1)} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix} = \mathbf{B} \begin{bmatrix} F_0 \\ F_1 \end{bmatrix},$$

where $\mathbf{B}$ is the 1×2 matrix $\mathbf{B} = \sum_{i=0}^{n/2-1} \begin{bmatrix} \alpha_{2i} & \alpha_{2i+1} \end{bmatrix} \mathbf{M}^{(1,2i+1)}$.

The computation of A is thus reduced to that of the matrix $\mathbf{B}$. Write $n' = n/2$. Following [22] and [15], we build the *subproduct tree* associated to the transition matrices $\mathbf{M}^{(j,i)}$. This is a complete binary tree having $d = \log_2(n) = \log_2(n')+1$ rows of nodes labeled as follows:

- the leaves of the tree are labeled by the matrices $\mathbf{L}^{(d-1,i)} = \mathbf{M}^{(2i+1,2i+3)}$, for $i = 0, \ldots, n'-1$;
- for $j = 0, \ldots, d-2$, there are 2^j nodes of depth j and the $(1+i)$-th one is indexed by the matrix $\mathbf{L}^{(j,i)} = \mathbf{L}^{(j+1,2i+1)}\mathbf{L}^{(j+1,2i)}$, for $0 \leq i \leq 2^j - 1$.

To estimate the degrees of the entries of $\mathbf{L}^{(j,i)}$, for $0 \leq u, v \leq 1$ and $j < d$ we define

$$\eta_{j,u,v} = 2^{d-j} - 2 + u + v. \tag{4}$$

Lemma 1 *For $0 \leq u, v \leq 1$ and $j = 0, \ldots, d-1$, the entry $L_{u,v}^{(j,i)}$ of $\mathbf{L}^{(j,i)}$ has degree at most $\eta_{j,u,v}$.*

Proof. This is done by a straightforward induction using the recursive definition of the matrices $\mathbf{L}^{(j,i)}$. □

$\underline{\mathsf{Expand}(\mathbf{a}, \mathbf{b}, \mathbf{c}, A, n)}$

Input: $A = \sum_{i=0}^{n-1} \alpha_i x^i$ and $\mathbf{a}, \mathbf{b}, \mathbf{c}$
Output: $\sum_{i=0}^{n-1} \alpha_i F_i$ expanded on the monomial basis

$\mathbf{L}^{(j,i)} \leftarrow \mathsf{SubproductTree}(\mathbf{a}, \mathbf{b}, \mathbf{c}, n)$
for $i = 0, \ldots, 2^{d-1} - 1$ **do**
  $v_0^{(d-1,i)} \leftarrow \alpha_{2i}$
  $v_1^{(d-1,i)} \leftarrow \alpha_{2i+1}$
for $j = d-2, \ldots, 0$ **do**
  for $i = 0, \ldots, 2^j - 1$ **do**
    $v_0^{(j,i)} \leftarrow v_0^{(j+1,2i)} + v_0^{(j+1,2i+1)} L_{0,0}^{(j+1,2i)} + v_1^{(j+1,2i+1)} L_{1,0}^{(j+1,2i)}$
    $v_1^{(j,i)} \leftarrow v_1^{(j+1,2i)} + v_0^{(j+1,2i+1)} L_{0,1}^{(j+1,2i)} + v_1^{(j+1,2i+1)} L_{1,1}^{(j+1,2i)}$
return $v_0^{(0,0)} F_0 + v_1^{(0,0)} F_1$

Fig. 1. Algorithm solving Problem Expand_n

Another induction shows that for $j = 0, \ldots, d-1$ and $i = 0, \ldots, 2^j - 1$, we have the equality

$$\mathbf{L}^{(j,i)} = \mathbf{M}^{(2^{d-j}i+1,\, 2^{d-j}(i+1)+1)}.$$

The cost of computing all matrices in the tree is $O(\mathsf{M}(n)\log(n))$, as in [11, Chap. 10]. Then, to compute $\mathbf{B}$, we go up the subproduct tree and perform linear combinations along the way: we maintain a family of 1×2 vectors $\mathbf{v}^{(j,i)} = [v_0^{(j,i)}\ v_1^{(j,i)}]$, with $j = 0, \ldots, d-1$ and $i = 0, \ldots, 2^j - 1$, such that

$$\mathbf{v}^{(d-1,i)} = [\alpha_{2i}\ \alpha_{2i+1}] \quad \text{and} \quad \mathbf{v}^{(j,i)} = \mathbf{v}^{(j+1,2i)} + \mathbf{v}^{(j+1,2i+1)} \mathbf{L}^{(j+1,2i)}. \tag{5}$$

The overall cost is again $O(\mathsf{M}(n)\log(n))$.

Remark that not all the nodes of the complete subproduct tree are actually needed in this algorithm. Indeed, its rightmost branch containing $\mathbf{L}^{(j,2^j-1)}$ for $0 \le j \le d-1$ is not necessary in the computation described in Equation (5).

In the pseudo-code in Figure 1, we make all scalar operations explicit, so as to make the transposition process easier in the next section. Starting from the sequences $\mathbf{a} = (a_1, \ldots, a_{n-1}), \mathbf{b} = (b_1, \ldots, b_{n-1}), \mathbf{c} = (c_1, \ldots, c_{n-1})$, the subroutine $\mathsf{SubproductTree}(\mathbf{a}, \mathbf{b}, \mathbf{c}, n)$ computes the matrices $\mathbf{M}^{(i,i+1)}$ for $0 \le i \le n-2$, then the matrices $\mathbf{L}^{(j,i)}$ for $1 \le j \le d-1$ and $0 \le i \le 2^j - 2$.

3 Transposed expansion

Let $r, s \ge 1$ and let $\mathbf{M}$ be a $r \times s$ matrix with entries in $\mathbb{K}$. The *transposition principle* [3, Th. 13.20] states that the existence of an algorithm for the

matrix-vector product $b \mapsto \mathbf{M}b$ implies the existence of an algorithm with the same cost, up to $O(r+s)$ operations, to perform the transposed matrix-vector product $c \mapsto \mathbf{M}^t c$. This section gives the transposed version of the conversion algorithm above: a similar algorithm is given in [7], but our derivation is more straightforward.

A fundamental operation is transposed polynomial multiplication. For k in $\mathbb{N}$, let $\mathbb{K}[x]_k$ be the $\mathbb{K}$-vector space of polynomials of degree less than k. Then, for B in $\mathbb{K}[x]$ of degree at most m, we let

$$\mathrm{mul}(., B, m, k) : \mathbb{K}[x]_k \to \mathbb{K}[x]_{k+m}$$

be the multiplication-by-B operator. The transpose of this map is denoted by $\mathrm{mul}^t(., B, m, k)$; by identifying $\mathbb{K}[x]_k$ with its dual, one sees that $\mathrm{mul}^t(., B, m, k)$ maps $\mathbb{K}[x]_{k+m}$ to $\mathbb{K}[x]_k$.

For a polynomial F of degree less than m, $m \geq 1$, let $\mathrm{rev}(F, m) = x^{m-1}F(1/x)$ denote the reversal of F. In [1,13], details of the transposed versions of plain, Karatsuba and FFT multiplications are given, with a cost matching that of the direct product. Without using such techniques, writing down the multiplication matrix shows that $\mathrm{mul}^t(., B, m, k)$ is

$$A \in \mathbb{K}[x]_{k+m} \mapsto (A\ \mathrm{rev}(B, m+1) \bmod x^{k+m}) \operatorname{div} x^m \in \mathbb{K}[x]_k.$$

Here, $F \operatorname{div} x^m$ denotes the quotient of F through the Euclidean division by x^m; formally, for $F = \sum_{i=0}^{k+m-1} f_i x^i$ in $\mathbb{K}[x]_{k+m}$, we have $F \operatorname{div} x^m = \sum_{i=0}^{k-1} f_{i+m} x^i$. Similarly, $F \bmod x^m$ denotes the remainder of F through the Euclidean division by x^m; this operation is thus a truncation in degree less than m.

Using standard multiplication algorithms, the above formulation leads to algorithms for the transposed product that are slower than those of [1,13]. However, here k and m are of the same order of magnitude, and only a constant factor is lost.

Using this tool, the transposed expansion algorithm in Figure 2 below is obtained by "reversing the flow" of the direct one in Figure 1. To derive it, we first need information on the degrees of the polynomials involved. For $j = 0, \ldots, d-1$, define

$$\delta_{j,0} = \max(1, 2^{d-j} - 2) \quad \text{and} \quad \delta_{j,1} = 2^{d-j} - 1.$$

Lemma 2 *For $j = 0, \ldots, d-1$ and $i = 0, \ldots, 2^j - 1$, the inequalities $\deg(v_0^{(j,i)}) < \delta_{j,0}$ and $\deg(v_1^{(j,i)}) < \delta_{j,1}$ hold.*

Proof. The proof is by decreasing induction on j; the case $j = d-1$ is clear, since $\delta_{d-1,0} = \delta_{d-1,1} = 1$. For $j < d-1$, supposing by induction that the claim

holds for index $j+1$, Equation (5) gives the inequalities

$$\deg(v_0^{(j,i)}) < \max\left(\delta_{j+1,0} + \deg(L_{0,0}^{(j+1,2i)}), \delta_{j+1,1} + \deg(L_{1,0}^{(j+1,2i)})\right)$$

and

$$\deg(v_1^{(j,i)}) < \max\left(\delta_{j+1,0} + \deg(L_{0,1}^{(j+1,2i)}), \delta_{j+1,1} + \deg(L_{1,1}^{(j+1,2i)})\right).$$

After a quick simplification, the definitions of the integers $\delta_{j,0}$ and $\delta_{j,1}$ and the degree bounds on the polynomials L given in Equation (4) and Lemma 1 prove the lemma. □

We are next going to describe the main loop of the direct algorithm in matrix terms. To do so, we introduce the following notation. For any positive integers a, b, $\mathbf{1}_{a,b}$ is the $a \times b$ matrix having 1's on the main diagonal and 0's everywhere else (note that this is a rectangular matrix). Similarly, $\mathbf{0}_{a,b}$ is the zero matrix of size $a \times b$. Further, for $j = 0, \ldots, d-1$ and $i = 0, \ldots, 2^j - 1$, and for $0 \le u, v \le 1$, let $\mathbf{t}_{u,v}^{(j,i)}$ be the $(\delta_{j,0} + \delta_{j,1}) \times (2\delta_{j+1,0} + 2\delta_{j+1,1})$ matrix of the operator

$$\mathrm{mul}(., L_{u,v}^{(j,i)}, \eta_{j,u,v} + 2 - 2u, \delta_{j,u}).$$

We can then define a matrix $\mathbf{T}^{(j,i)}$ given in block format by

$$\begin{bmatrix} \mathbf{1}_{\delta_{j,0},\delta_{j+1,0}} & \mathbf{0}_{\delta_{j,0},\delta_{j+1,1}} & \mathbf{t}_{0,0}^{(j+1,2i)} & \mathbf{t}_{1,0}^{(j+1,2i)} \\ \mathbf{0}_{\delta_{j,1},\delta_{j+1,0}} & \mathbf{1}_{\delta_{j,1},\delta_{j+1,1}} & \mathbf{t}_{0,1}^{(j+1,2i)} & \mathbf{t}_{1,1}^{(j+1,2i)} \end{bmatrix}.$$

Let $\mathbf{c}^{(j,i)}$ be the sequence of the coefficients of the polynomial $v_0^{(j,i)}$, up to degree $\delta_{j,0} - 1$, followed by those of $v_1^{(j,i)}$, taken up to degree $\delta_{j,1} - 1$. Hence, $\mathbf{c}^{(j,i)}$ has length $\delta_{j,0} + \delta_{j,1}$. The equality $\mathbf{v}^{(j,i)} = \mathbf{v}^{(j+1,2i)} + \mathbf{v}^{(j+1,2i+1)}\mathbf{L}^{(j+1,2i)}$ can now be rewritten as

$$\left[\mathbf{c}^{(j,i)}\right] = \mathbf{T}^{(j,i)} \begin{bmatrix} \mathbf{c}^{(j+1,2i)} \\ \mathbf{c}^{(j+1,2i+1)} \end{bmatrix}. \tag{6}$$

Finally, let $\mathbf{c}^{(j)}$ be the sequence obtained by concatenating $\mathbf{c}^{(j,0)}, \ldots, \mathbf{c}^{(j,2^j-1)}$: this is the sequence of all coefficients of $v_0^{(j,0)}, \ldots, v_1^{(j,2^j-1)}$. Let $\mathbf{T}^{(j)}$ be the block diagonal matrix having $\mathbf{T}^{(j,0)}, \ldots, \mathbf{T}^{(j,2^j-1)}$ on the diagonal; then Equation (6) yields

$$\left[\mathbf{c}^{(j)}\right] = \mathbf{T}^{(j)} \left[\mathbf{c}^{(j+1)}\right].$$

After initialization of $v_0^{(d-1,0)}, \ldots, v_1^{(d-1,2^{d-1}-1)}$, that is, of $\mathbf{c}^{(d-1)}$, the algorithm of Figure 1 can be interpreted as follows: it computes the coefficient sequences $\mathbf{c}^{(d-2)}, \ldots, \mathbf{c}^{(0)}$ by application of the matrices $\mathbf{T}^{(d-2)}, \ldots, \mathbf{T}^{(0)}$. The transposed algorithm will thus successively apply the *transposed matrices* of

$\mathbf{T}^{(0)}, \ldots, \mathbf{T}^{(d-2)}$ to suitable vectors, due to the equality

$$\left(\mathbf{T}^{(0)} \cdots \mathbf{T}^{(d-2)}\right)^t = \mathbf{T}^{(d-2)^t} \cdots \mathbf{T}^{(0)^t}.$$

Since we identify a vector space of the form $\mathbb{K}[x]_k$ with its dual, at the entry of step j of the transposed algorithm, we are still given polynomials, written $v_0^{(j,0)}, v_1^{(j,0)}, \ldots, v_0^{(j,2^j-1)}, v_1^{(j,2^j-1)}$. As before, we let $\mathbf{c}^{(j)}$ be the sequence obtained by taking all coefficients of the polynomials $v_0^{(j,i)}$ and $v_1^{(j,i)}$, up to degrees respectively $\delta_{j,0}-1$ for the polynomials $v_0^{(j,i)}$ and $\delta_{j,1}-1$ for the polynomials $v_1^{(j,i)}$. Then, the main loop of the transposed algorithm succesively computes

$$\left[\mathbf{c}^{(j+1)}\right] = \mathbf{T}^{(j)^t} \left[\mathbf{c}^{(j)}\right].$$

Similarly to the direct algorithm, we wish to describe this operation in polynomial terms. By construction, the transposed matrix of $\mathbf{T}^{(j)}$ is block-diagonal, with blocks of the form

$$\begin{bmatrix} \mathbf{1}_{\delta_{j+1,0},\delta_{j,0}} & \mathbf{0}_{\delta_{j+1,0},\delta_{j,1}} \\ \mathbf{0}_{\delta_{j+1,1},\delta_{j,0}} & \mathbf{1}_{\delta_{j+1,1},\delta_{j,1}} \\ \mathbf{t}_{0,0}^{(j+1,2i)^t} & \mathbf{t}_{0,1}^{(j+1,2i)^t} \\ \mathbf{t}_{1,0}^{(j+1,2i)^t} & \mathbf{t}_{1,1}^{(j+1,2i)^t} \end{bmatrix}$$

on the diagonal. In polynomial terms, the action of this block on the coefficient vector of $[v_0^{(j,i)} v_1^{(j,i)}]$ is obtained as follows:

- The first component of the result is obtained by applying $\mathbf{1}_{\delta_{j+1,0},\delta_{j,0}}$ to the coefficient vector of $v_0^{(j,i)}$. In polynomial terms, this is the truncation $v_0^{(j,i)} \bmod x^{\delta_{j+1,0}}$.
- Similarly, the second component of the result is $v_1^{(j,i)} \bmod x^{\delta_{j+1,1}}$.
- The third component is obtained as the sum of two terms; both of them are transposed polynomial multiplications, of respectively $v_0^{(j,i)}$ with $L_{0,0}^{(j+1,2i)}$ and $v_1^{(j,i)}$ with $L_{0,1}^{(j+1,2i)}$.
- Similarly, the last component is obtained as the sum of two transposed polynomial multiplications, this time with $L_{1,0}^{(j+1,2i)}$ and $L_{1,1}^{(j+1,2i)}$.

The former discussion justifies the main loop of the algorithm of Figure 2; remark that the order of the inner loop $i = 2^j - 1, \ldots, 0$ is irrelevant: we could keep the increasing order of the initial algorithm. To complete the derivation of the transposed algorithm, we must describe the initialization and final step.

- The original algorithm initializes the vector $v_0^{(d-1,i)}$ with the coefficient α_{2i} of A and $v_1^{(d-1,i)}$ with the coefficient α_{2i+1} of A. In the transposed algorithm,

$\underline{\mathsf{Expand}^t(\mathbf{a},\mathbf{b},\mathbf{c},A,n)}$

Input: $A=\sum_{i=0}^{n-1}\alpha_i x^i$ and $\mathbf{a},\mathbf{b},\mathbf{c}$
Output: the polynomial of coefficients $(\sum_{i=0}^{n-1}\alpha_i\mathrm{coeff}(F_j,i))_{j<n}$

$\mathbf{L}^{(j,i)} \leftarrow \mathsf{SubproductTree}(\mathbf{a},\mathbf{b},\mathbf{c},n)$
$v_0^{(0,0)} \leftarrow \mathrm{mul}^t(A,F_0,0,\delta_{0,0})$
$v_1^{(0,0)} \leftarrow \mathrm{mul}^t(A,F_1,1,\delta_{0,1})$
for $j=0,\ldots,d-2$ **do**
  for $i=2^j-1,\ldots,0$ **do**
    $v_0^{(j+1,2i)} \leftarrow v_0^{(j,i)} \bmod x^{\delta_{j+1,0}}$
    $v_1^{(j+1,2i)} \leftarrow v_1^{(j,i)} \bmod x^{\delta_{j+1,1}}$
    $v_0^{(j+1,2i+1)} \leftarrow \mathrm{mul}^t(v_0^{(j,i)},L_{0,0}^{(j+1,2i)},\eta_{j+1,0,0},\delta_{j+1,0}) + \mathrm{mul}^t(v_1^{(j,i)},L_{0,1}^{(j+1,2i)},\eta_{j+1,0,1},\delta_{j+1,0})$
    $v_1^{(j+1,2i+1)} \leftarrow \mathrm{mul}^t(v_0^{(j,i)},L_{1,0}^{(j+1,2i)},\eta_{j+1,1,0},\delta_{j+1,1}) + \mathrm{mul}^t(v_1^{(j,i)},L_{1,1}^{(j+1,2i)},\eta_{j+1,1,1},\delta_{j+1,1})$
return $v_0^{(d-1,0)}+v_1^{(d-1,0)}x+\cdots+v_1^{(d-1,2^{d-1}-1)}x^{n-1}$

Fig. 2. Algorithm solving Problem Expand_n^t

the final polynomials $v_0^{(d-1,i)}$ and $v_1^{(d-1,i)}$ have degrees less than 1, they are thus actually constants. Then, the transposed operation returns the polynomial having $v_0^{(d-1,0)}, v_1^{(d-1,0)}, \ldots, v_0^{(d-1,2^{d-1}-1)}, v_1^{(d-1,2^{d-1}-1)}$ as coefficients.

- The original algorithm returns $v_0^{(0,0)}F_0+v_1^{(0,0)}F_1$, where F_0 has degree 0 and F_1 degree 1. As before, one can rewrite this operation in matrix terms, as applying the operator

$$\begin{bmatrix} \mathrm{mul}(.,\,F_0,0,\,\delta_{0,0}) & \mathrm{mul}(.,\,F_1,1,\,\delta_{0,1}) \end{bmatrix}$$

to the coefficient vector of $[v_0^{(0,0)}v_1^{(0,0)}]$. Since $v_0^{(0,0)}$ and $v_1^{(0,0)}$ have degrees less than respectively $\delta_{0,0}$ and $\delta_{0,1}$, the transposed operation computes $v_0^{(0,0)}$ by applying $\mathrm{mul}^t(.,F_0,0,\delta_{0,0})$ and $v_1^{(0,0)}$ by applying $\mathrm{mul}^t(.,F_1,1,\delta_{0,1})$; it takes place at the beginning of Figure 2.

4 Decomposition Problem

The Favard-Shohat theorem [8,21], see also [5, Th. 4.4], asserts that for (F_i) as in (1), there exists a linear form $L:\mathbb{K}[x]\to\mathbb{K}$ for which (F_i) is *formally*

orthogonal, in the sense that, for $i \geq 1$,

$$L(F_i F_j) = 0 \quad \text{for} \quad 0 \leq j < i, \quad L(F_i^2) \neq 0.$$

The linear form L is specified by its *moments* $L(x^i)$, for $i \geq 0$, or equivalently by the generating series

$$S = \sum_{i \geq 0} L(x^i) x^i \in \mathbb{K}[[x]].$$

For completeness, we give in the following theorem a self-contained, constructive, proof of this classical result, showing how to compute truncations of S. The proof is inspired by the presentation in [9, §3].

To simplify notation, we use the customary convention that products are equal to 1 whenever their upper indexes are smaller than the lower ones.

Theorem 2 *Let (F_i) be the sequence satisfying $F_{-1} = 0, F_0 = 1$ and recurrence (1). Define the sequence (G_i) by $G_{-1} = 0$, $G_0 = 1$ and, for $i \geq 1$*

$$G_i = (a_{i+1}x + b_{i+1})G_{i-1} + c_{i+1}G_{i-2}.$$

Then, there exists a $\mathbb{K}$-linear form $L : \mathbb{K}[x] \to \mathbb{K}$ such that

$$L(F_i F_j) = 0 \quad \textit{for} \quad i \neq j, \quad \textit{and} \quad L(F_i^2) = (-1)^i \frac{c_2 \cdots c_{i+1}}{a_{i+1}} \quad \textit{for} \quad i \geq 0. \tag{7}$$

Moreover, for any $i \geq 1$, the following equality holds between truncated series in $\mathbb{K}[[x]]$:

$$\frac{\operatorname{rev}(G_{i-1}, i)}{\operatorname{rev}(F_i, i+1)} = \sum_{j \geq 0} L(x^j) x^j \mod x^{2i}. \tag{8}$$

Proof. For $i \geq 0$, write $F_i^\star = \operatorname{rev}(F_i, i+1)$ and $G_i^\star = \operatorname{rev}(G_i, i+1)$, so that in particular $F_0^* = G_0^* = 1$; define also $F_{-1}^\star = G_{-1}^\star = 0$. These polynomials satisfy the recurrences

$$F_i^\star = (a_i + b_i x)F_{i-1}^\star + c_i x^2 F_{i-2}^\star, \quad G_i^\star = (a_{i+1} + b_{i+1}x)G_{i-1}^\star + c_{i+1}x^2 G_{i-2}^\star,$$

for $i \geq 1$, which can be recast into the matrix form

$$\begin{bmatrix} F_i^\star & G_{i-1}^\star \\ F_{i+1}^\star & G_i^\star \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ c_{i+1}x^2 & (a_{i+1} + b_{i+1}x) \end{bmatrix} \begin{bmatrix} F_{i-1}^\star & G_{i-2}^\star \\ F_i^\star & G_{i-1}^\star \end{bmatrix}.$$

Taking determinants, we deduce that for $i \geq 1$ the following identity holds

$$\frac{G_i^\star}{F_{i+1}^\star} - \frac{G_{i-1}^\star}{F_i^\star} = -c_{i+1}\frac{F_{i-1}^\star}{F_{i+1}^\star} x^2 \left(\frac{G_{i-1}^\star}{F_i^\star} - \frac{G_{i-2}^\star}{F_{i-1}^\star} \right).$$

$\underline{\mathsf{Decomp}(\mathbf{a}, \mathbf{b}, \mathbf{c}, A, n)}$

Input: $A = \sum_{i=0}^{n-1} u_i x^i$ and $\mathbf{a}, \mathbf{b}, \mathbf{c}$
Output: $\alpha_0, \ldots, \alpha_{n-1}$ such that $A = \sum_{i=0}^{n-1} \alpha_i F_i$

$\mathbf{a}' \leftarrow \mathsf{cat}(\mathbf{a}, 1)$
$\mathbf{b}' \leftarrow \mathsf{cat}(\mathbf{b}, 1)$
$\mathbf{c}' \leftarrow \mathsf{cat}(\mathbf{c}, 1)$
$F \leftarrow \mathsf{Expand}(\mathbf{a}', \mathbf{b}, \mathbf{c}, x^n, n+1)$
$G \leftarrow \mathsf{Expand}(\mathbf{Sa}', \mathbf{Sb}', \mathbf{Sc}', x^{n-1}, n)$
$Q \leftarrow \mathrm{rev}(G, n)/\mathrm{rev}(F, n+1) \mod x^{2n-1}$
$V \leftarrow \mathrm{mul}^t(Q, A, n-1, n)$
$w \leftarrow \mathsf{Expand}_n^t(\mathbf{a}, \mathbf{b}, \mathbf{c}, V, n)$
$d_i \leftarrow (-1)^i c_2 \cdots c_{i+1}/a_{i+1}$, for $0 \le i < n$
$\alpha_i \leftarrow w_i/d_i$ for $0 \le i < n$, where $w = [w_0, \ldots, w_{n-1}]^t$
return $\alpha_0, \ldots, \alpha_{n-1}$

Fig. 3. Algorithm solving Problem Decomp_n

Applying it to $i, i-1, \ldots$ and denoting $\gamma_i = c_2 \cdots c_i$, we get that for all $i \ge 1$,

$$\frac{G_i^\star}{F_{i+1}^\star} - \frac{G_{i-1}^\star}{F_i^\star} = (-1)^i \frac{\gamma_{i+1}}{F_i^\star F_{i+1}^\star} x^{2i}. \tag{9}$$

With $\gamma_1 = 1$ as per our convention, a separate check shows that Equation (9) also holds for $i = 0$.

For $i \ge 0$, $F_i^\star$ has constant coefficient $\delta_i = a_1 \cdots a_i$, which is non-zero, and is thus invertible in $\mathbb{K}[[x]]$. Since the γ_{i+1} are non-zero as well, Equation (9) shows that the sequence $G_i^\star/F_{i+1}^\star$ is Cauchy and thus convergent in $\mathbb{K}[[x]]$. Besides, if we let S be its limit, summing up Equation (9) for $i, i+1, \ldots$ yields

$$S = \frac{G_{i-1}^\star}{F_i^\star} + (-1)^i \frac{\gamma_{i+1}}{\delta_i \delta_{i+1}} x^{2i} \mod x^{2i+1}, \quad \text{for} \quad i \ge 0. \tag{10}$$

Write $S = \sum_{i \ge 0} \ell_i x^i$ and define the linear form L on $\mathbb{K}[x]$ by $L(x^i) = \ell_i$. Then Equation (8) is a direct consequence of (10).

Expanding the product shows that for any $i, j \in \mathbb{N}$, the coefficient of x^{i+j} in $SF_i^\star$ equals $L(F_i x^j)$. Thus, for $i \ge 0$, multiplying Equation (10) by $F_i^\star$ and equating coefficients of $x^i, \ldots, x^{2i-1}$ and x^{2i} implies $L(F_i x^j) = 0$ for $j < i$ and $L(F_i x^i) = (-1)^i \gamma_{i+1}/\delta_{i+1}$. By linearity, this shows that L also satisfies Equality (7). $\square$

Proof of Theorem 1. We can now prove the second part of Theorem 1, dealing with expansions in the monomial basis. The corresponding algorithm is given in Figure 3.

We first compute $(L(x^i))_{i<2n-1}$. To do this, we start from the sequences $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ to which we add the element 1, in order to make the polynomial $F = F_n$ well-defined (any non-zero choice would do). We then use the algorithm Expand of Section 2 to compute $G = G_{n-1}$ and $F = F_n$ and we determine the power series expansion $\mathrm{rev}(G_{n-1}, n)/\mathrm{rev}(F_n, n+1) \bmod x^{2n-1}$. The first step takes $O(\mathsf{M}(n)\log(n))$ operations, while the second one takes $O(\mathsf{M}(n))$, either using Newton iteration [11, Chap. 9], or a fast algorithm for inverting a non-singular triangular Toeplitz matrix, cf. [6] and [18, §2.5]. In the pseudo-code we use the notation $\mathbf{Sx}$ for the shifted sequence (x_{i+1}) of $\mathbf{x} = (x_i)$ and the notation cat for concatenation.

Consider finally the matrix $\mathbf{F}_n = [F_{i,j}]_{0\le i,j<n}$ defined in the introduction, and let $\mathbf{H}_n = [H_{i,j}]_{0\le i,j<n}$ be the $n \times n$ Hankel matrix with $H_{i,j} = L(x^{i+j})$. Let next $\mathbf{D}_n$ be the diagonal matrix of size n, whose i-th diagonal entry is $L(F_i^2) \neq 0$. For $0 \le i, j < n$, the (i,j)-entry of the matrix $\mathbf{F}_n^t\,\mathbf{H}_n\,\mathbf{F}_n$ equals $\sum_{s,t=0}^{n-1} F_{s,i} L(x^{s+t}) F_{t,j}$, which is $L(F_i F_j)$; hence, this is zero for $i \neq j$ and $L(F_i^2)$ otherwise. We deduce the factorization

$$\mathbf{F}_n^t\,\mathbf{H}_n\,\mathbf{F}_n \;=\; \mathbf{D}_n, \quad \text{or} \quad \mathbf{F}_n^{-1} \;=\; \mathbf{D}_n^{-1}\,\mathbf{F}_n^t\,\mathbf{H}_n.$$

Equation (7) shows that one can compute the entries of $\mathbf{D}_n$ in $O(n)$ operations.

At this stage, all elements of $\mathbf{D}_n$ and $\mathbf{H}_n$ are known. Right-multiplication of $\mathbf{H}_n$ by the coefficient vector of a polynomial $A \in \mathbb{K}[x]_n$ amounts to the transposed multiplication of the polynomial $Q = \sum_{i=0}^{2n-2} L(x^i)x^i$ by A, that can be performed in time $\mathsf{M}(n)+O(n)$. Remark that if one allows Fast Fourier Transform, this is a classical result [23, Pb. 4.2.3]; however, for more general polynomial multiplication models, it is unknown how to obtain a $\mathsf{M}(n)+O(n)$ cost without using transposed multiplication [13].

Using the transposed expansion algorithm Expand^t of the previous section, multiplication by $\mathbf{F}_n^t$ costs $O(\mathsf{M}(n)\log(n))$. Finally, multiplying by $\mathbf{D}_n^{-1}$ takes linear time. This concludes the proof of Theorem 1. □

Acknowledgments. We thank an anonymous referee for several useful remarks. This work was supported in part by the French National Agency for Research (ANR Project "Gecko") and the Microsoft Research-INRIA Joint Centre. The third author acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada Research Chairs Program and MITACS.

References

[1] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM, 2003.

[2] A. Bostan, B. Salvy, and É. Schost. Power series composition and change of basis. In *ISSAC'08*, pages 269–276. ACM, 2008.

[3] P. Bürgisser, M. Clausen, and A. M. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren Math. Wiss.* Springer–Verlag, 1997.

[4] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[5] T. S. Chihara. *An introduction to orthogonal polynomials*. Gordon and Breach Science Publishers, New York, 1978. Mathematics and its Applications, Vol. 13.

[6] D. Commenges and M. Monsion. Fast inversion of triangular Toeplitz matrices. *IEEE Trans. Automat. Control*, 29(3):250–251, 1984.

[7] J. R. Driscoll, D. M. Healy, Jr., and D. N. Rockmore. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. *SIAM J. Comp.*, 26(4):1066–1099, 1997.

[8] J. Favard. Sur les polynômes de Tchebicheff. *Comptes-Rendus de l'Académie des Sciences*, 200:2052–2053, 1935.

[9] P. Flajolet. Combinatorial aspects of continued fractions. *Discrete Math.*, 32(2):125–161, 1980.

[10] M. Frumkin. A fast algorithm for expansion over spherical harmonics. *Appl. Algebra Engrg. Comm. Comput.*, 6(6):333–343, 1995.

[11] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.

[12] J. Gerhard. Modular algorithms for polynomial basis conversion and greatest factorial factorization. In *RWCA'00*, pages 125–141, 2000.

[13] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm, I. *Appl. Algebra Engrg. Comm. Comp.*, 14(6):415–438, 2004.

[14] G. Heinig. Fast and superfast algorithms for Hankel-like matrices related to orthogonal polynomials. In *NAA'00*, volume 1988 of *LNCS*, pages 361–380. Springer-Verlag, 2001.

[15] P. M. Kogge and H. S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, 22:786–793, 1973.

[16] G. Leibon, D. Rockmore, and G. Chirikjian. A fast Hermite transform with applications to protein structure determination. In *SNC'07*, pages 117–124. ACM, 2007.

[17] V. Y. Pan. New fast algorithms for polynomial interpolation and evaluation on the Chebyshev node set. *Computers and Mathematics with Applications*, 35(3):125–129, 1998.

[18] Victor Y. Pan. *Structured matrices and polynomials*. Birkhäuser Boston Inc., Boston, MA, 2001. Unified superfast algorithms.

[19] D. Potts, G. Steidl, and M. Tasche. Fast algorithms for discrete polynomial transforms. *Math. Comp.*, 67(224):1577–1590, 1998.

[20] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[21] J. Shohat. The relation of the classical orthogonal polynomials to the polynomials of Appell. *Amer. J. Math.*, 58(3):453–464, 1936.

[22] H. S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM*, 20:27–38, 1973.

[23] C. Van Loan. *Computational frameworks for the fast Fourier transform*, volume 10 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.