

Fast Conversion Algorithms for Orthogonal Polynomials

Alin Bostan Bruno Salvy

*Algorithms Project, INRIA 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.

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

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

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 $\mathbb{K}[x]$.

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

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 (\mathbf{Expand}_n). Given $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{K}$, compute the coefficients on the monomial basis of the polynomial A defined by the map

$$[\alpha_0, \dots, \alpha_{n-1}] \mapsto A = \sum_{i=0}^{n-1} \alpha_i F_i \quad (2)$$

Decomposition Problem (\mathbf{Decomp}_n). Conversely, given the coefficients of A on the monomial basis, recover the coefficients $\alpha_0, \dots, \alpha_{n-1}$ 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 \mathbf{Expand}_n amounts to multiplying the matrix \mathbf{F}_n by the vector $[\alpha_0, \dots, \alpha_{n-1}]^t$; hence, the inverse map \mathbf{Decomp}_n is well-defined, since \mathbf{F}_n is an upper-triangular matrix whose i -th diagonal entry $F_{i,i} = a_1 a_2 \cdots a_i$ is non-zero. As we will see, the *dual problem* (multiplying the matrix \mathbf{F}_n^t by a vector), denoted \mathbf{Expand}_n^t , plays an important role as well.

Naive algorithms work in complexity $O(n^2)$ for both problems \mathbf{Expand}_n and \mathbf{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 \mathbf{Decomp}_n .

As usual, we denote by \mathbf{M} a *multiplication time* function, such that polynomials of degree less than n in $\mathbb{K}[x]$ can be multiplied in $\mathbf{M}(n)$ operations in \mathbb{K} , when written in the monomial basis. Besides, we impose the usual super-linearity conditions of [10, Chap. 8]. Using Fast Fourier Transform algorithms, $\mathbf{M}(n)$ 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 [18,4].

Theorem 1 *Problems \mathbf{Expand}_n and \mathbf{Decomp}_n can be solved in $O(\mathbf{M}(n) \log(n))$ 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 \mathbf{Expand}_n . From these, one could readily infer fast algorithms for Problem \mathbf{Expand}_n itself.

In [17], the question is the computation of the values

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

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 [6] studies the transposed problem: computing the map

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

The algorithm in [6] is (roughly, see [17] for details) the transpose of the one in [17]: 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 $O(\mathbf{M}(n) \log(n))$ algorithm has appeared before, except for particular families of polynomials, like Legendre [9], Chebyshev [16] and Hermite [15]. In the case of arbitrary orthogonal polynomials, the best complexity result we are aware of is due to Heinig [13], who gives a $O(\mathbf{M}(n) \log^2(n))$ 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 [17], one sees mentions of left and right inverses for the related problem

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

In [15], 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 $O(\mathbf{M}(n) \log(n))$ or $O(\mathbf{M}(n) \log^2(n))$. In [2], we give algorithms of lower cost $O(\mathbf{M}(n))$ for many classical orthogonal polynomials (Jacobi, Hermite, Laguerre, ...), for both Problems Expand_n and Decomp_n .

Main ideas. Here is a brief description of the strategy used to obtain the complexity estimate of Theorem 1. The complete treatment with detailed algorithms is given in Sections 2 and 3. Three main ingredients are used: (i) a $O(\mathbf{M}(n) \log(n))$ algorithm for Problem Expand_n ; (ii) the transposition principle; (iii) the Favard-Shohat theorem.

We first recast (1) into the matrix recurrence $[F_i, F_{i+1}]^t = \mathbf{M}^{(i)}(x)[F_{i-1}, F_i]^t$, where $\mathbf{M}^{(i)}$ is a 2×2 polynomial matrix. Problem Expand_n then amounts to

computing $T = \alpha_0 \mathbf{M}^{(0)} + \alpha_1 \mathbf{M}^{(1)} \mathbf{M}^{(0)} + \dots + \alpha_{n-1} \mathbf{M}^{(n-1)} \dots \mathbf{M}^{(0)}$. This is done by using a divide-and-conquer algorithm similar to the one in [11, Th. 2.4] for the conversions between Newton and monomial bases. Assuming for simplicity that n is even, we rely on the decomposition $T = T_0 + T_1 \mathbf{M}^{(\frac{n}{2}-1)} \dots \mathbf{M}^{(0)}$, with

$$\begin{aligned} T_0 &= \alpha_0 \mathbf{M}^{(0)} + \dots + \alpha_{\frac{n}{2}-1} \mathbf{M}^{(\frac{n}{2}-1)} \dots \mathbf{M}^{(0)} \\ T_1 &= \alpha_{\frac{n}{2}} \mathbf{M}^{(\frac{n}{2})} + \dots + \alpha_{n-1} \mathbf{M}^{(n-1)} \dots \mathbf{M}^{(\frac{n}{2})}. \end{aligned}$$

In Section 2, a slightly different but more efficient version of this algorithm is given.

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

Then, the Favard-Shohat theorem [7,19] ensures the existence of an inner product $\langle \cdot, \cdot \rangle$ on the space $\mathbb{K}[x]$ with respect to which the sequence $(F_n)_n$ is an orthogonal basis. This implies the matrix equality $\mathbf{F}_n^t \mathbf{H}_n \mathbf{F}_n = \mathbf{D}_n$, where $\mathbf{H}_n = [h_{i,j}]_{0 \leq i,j < n}$ is the Gram matrix with $h_{i,j} = \langle x^i, x^j \rangle$ and \mathbf{D}_n is an invertible diagonal matrix. Its equivalent form $\mathbf{F}_n^{-1} = \mathbf{D}_n^{-1} \mathbf{F}_n^t \mathbf{H}_n$ shows that, once \mathbf{H}_n and \mathbf{D}_n are determined, Problem Decomp_n amounts to the computation of the map $w \in \mathbb{K}^n \mapsto \mathbf{F}_n^t w \in \mathbb{K}^n$, that is, to solving Expand_n^t . Finally, a constructive version of the Favard-Shohat theorem shows that determining the Gram matrix \mathbf{H}_n can be reduced to two instances of Problem Expand_n .

In summary, by the Favard-Shohat theorem, Decomp_n is reduced to Expand_n and Expand_n^t , which can be solved in $O(\mathbf{M}(n) \log(n))$, by a direct divide-and-conquer algorithm for the first and the transposition principle for the second.

2 Expansion Problem

We first describe the conversion from the orthogonal basis to the monomial one, and its transpose. The content of this section is mostly already known. However, our algorithm for the inverse operation rests crucially on these conversions, so we prefer to make them explicit.

In the following, we always suppose for simplicity that the number of unknown coefficients n is a power of two. For a polynomial F of degree less than m , $m \geq 1$, we denote by $\text{rev}(F, m) = x^{m-1} F(1/x)$ the reversal of F .

Expansion from an orthogonal basis. Given $\alpha_0, \dots, \alpha_{n-1}$, we compute here the expansion on the monomial basis of

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

The ideas are classical; our presentation is taken from [17]. However, our use of “classical” fast multiplication techniques avoids the need of precomputed constants arising in [17], 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 $\mathbf{M}^{(i,j)} = \mathbf{M}^{(j-1,j)}\mathbf{M}^{(j-2,j-1)} \dots \mathbf{M}^{(i,i+1)}$; 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 rewrite A as

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

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

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

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 [20] and [14], 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 $i = 0, \dots, n' - 1$;

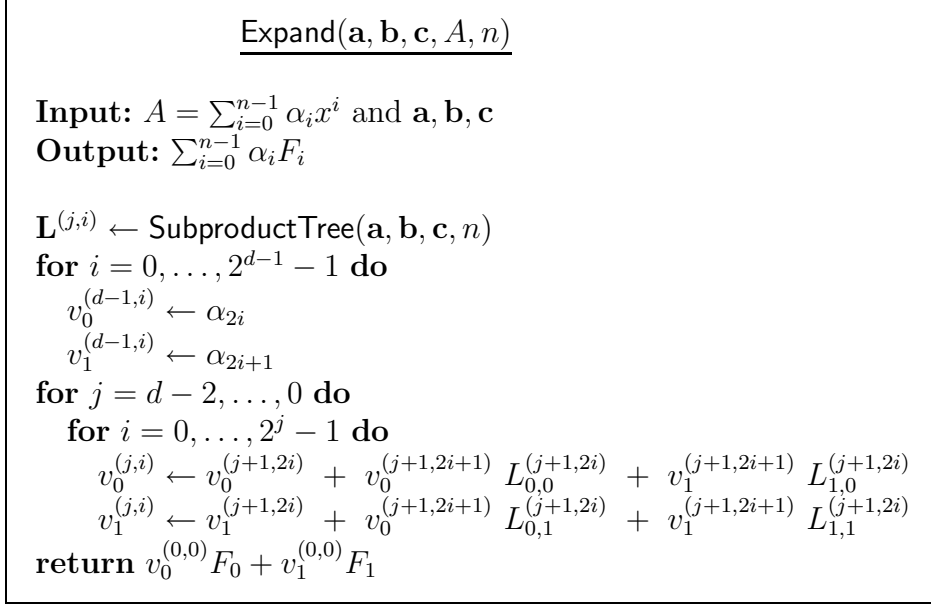


Fig. 1. Algorithm solving Problem Expand_n

- for $j = 0, \dots, d - 2$, 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 $0 \leq i \leq 2^j - 1$.

The entries $L_{u,v}^{(j,i)}$ of $\mathbf{L}^{(j,i)}$ have degrees at most $2^{d-j} - 2 + u + v$, with $0 \leq u, v \leq 1$. An easy induction also shows that for $j = 0, \dots, d - 1$ and $i = 0, \dots, 2^j - 1$, 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 $O(\mathbf{M}(n) \log(n))$, as in [10, Chapter 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 $j = 0, \dots, d - 1$ and $i = 0, \dots, 2^j - 1$, such that

$$\mathbf{v}^{(d-1,i)} = [\alpha_{2i} \ \alpha_{2i+1}] \quad \text{and} \quad \mathbf{v}^{(j,i)} = \mathbf{v}^{(j+1,2i)} + \mathbf{v}^{(j+1,2i+1)} \mathbf{L}^{(j+1,2i)}. \quad (4)$$

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

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 $0 \leq j \leq d - 1$ is not necessary in the computation described in Equation (4).

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

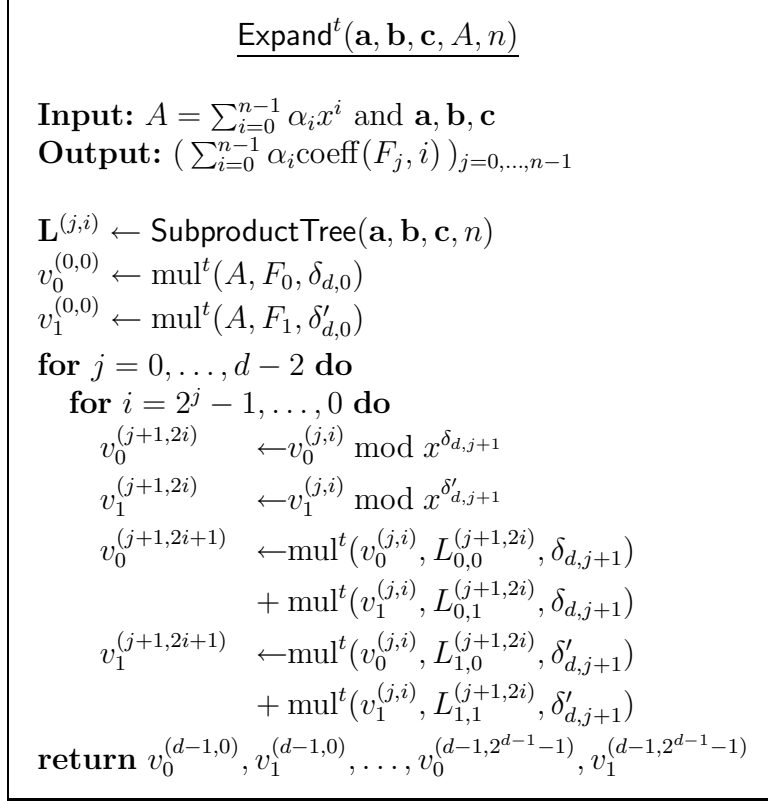


Fig. 2. Algorithm solving Problem Expand^t_n

Transposed expansion. Let $r, s \geq 1$ 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 paragraph gives the transposed version of the conversion algorithm above: a similar algorithm is given in [6], but our derivation is substantially more compact.

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 m , we let $\text{mul}(\cdot, B, k)$ be the multiplication-by- B operator, defined over $\mathbb{K}[x]_k$; its image lies in $\mathbb{K}[x]_{k+m}$.

The transpose of this map is denoted by $\text{mul}^t(\cdot, B, k)$; by identifying $\mathbb{K}[x]_k$ with its dual, one sees that $\text{mul}^t(\cdot, B, k)$ maps $\mathbb{K}[x]_{k+m}$ to $\mathbb{K}[x]_k$. In [1,12], 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 $\text{mul}^t(\cdot, B, k)$ is

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

Using standard multiplication algorithms, this formulation leads to slower algorithms than those of [1,12]. 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 is obtained by reversing the flow of the direct one in Figure 1. The loops are traversed in opposite order. Then, the operation $\mathbf{v}^{(j,i)} = \mathbf{v}^{(j+1,2i)} + \mathbf{v}^{(j+1,2i+1)}\mathbf{L}^{(j+1,2i)}$ in the inner loop is replaced by a truncated copy of $\mathbf{v}^{(j,i)}$ into $\mathbf{v}^{(j+1,2i)}$ and a transposed matrix-vector product, where polynomial multiplications are replaced by transposed multiplications. To perform truncations and transposed multiplications, we need information on the degrees of the polynomials involved. By induction, we get the following inequalities, for $j = 0, \dots, d-1$ and $i = 0, \dots, 2^j - 1$,

$$\deg(v_0^{(j,i)}) < \delta_{d,j} = \max(0, 2^{d-j} - 3) + 1, \quad \deg(v_1^{(j,i)}) < \delta'_{d,j} = 2^{d-j} - 1.$$

This information enables us to write the transposed algorithm in Figure 2. Using either the transposition principle or a direct analysis, one sees that the cost of this algorithm is $O(M(n) \log(n))$.

3 Decomposition Problem

The Favard-Shohat theorem [7,19], see also [5, Theorem 4.4], asserts that for (F_i) as in (1), there exists a linear form $L : \mathbb{K}[x] \rightarrow \mathbb{K}$ for which (F_i) is *formally orthogonal*, in the sense that, for $i \geq 1$,

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

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

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

For completeness, we give in the following theorem a self-contained, constructive, proof of this classical result, showing how to compute truncations of S_L . The proof is inspired by the presentation in [8, Section 3].

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 $L : \mathbb{K}[x] \rightarrow \mathbb{K}$ such that

$$L(F_i F_j) = 0 \quad \text{for } i \neq j, \quad \text{and} \quad L(F_i^2) = (-1)^i \frac{c_2 \cdots c_{i+1}}{a_{i+1}} \quad \text{for } i \geq 0. \quad (5)$$

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

$$\frac{\text{rev}(G_{i-1}, i)}{\text{rev}(F_i, i+1)} = \sum_{i \geq 0} L(x^i)x^i \pmod{x^{2i}}. \quad (6)$$

PROOF. For $i \geq 0$, write $F_i^* = \text{rev}(F_i, i+1)$ and $G_i^* = \text{rev}(G_i, i+1)$. Let also define $F_{-1}^* = G_{-1}^* = 0$. These polynomials satisfy the recurrences

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

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

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

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

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

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

$$\frac{G_i^*}{F_{i+1}^*} - \frac{G_{i-1}^*}{F_i^*} = (-1)^i \frac{\gamma_{i+1}}{F_i^* F_{i+1}^*} x^{2i}. \quad (7)$$

A separate check shows that Equation (7) also holds for $i = 0$.

For $i \geq 0$, F_i^* 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 (7) shows that the sequence G_i^*/F_{i+1}^* is Cauchy and thus convergent in $\mathbb{K}[[x]]$. Besides, if we let S be its limit, summing up Equation (7) for $i, i+1, \dots$ yields

$$S = \frac{G_{i-1}^*}{F_i^*} + (-1)^i \frac{\gamma_{i+1}}{\delta_i \delta_{i+1}} x^{2i} \pmod{x^{2i+1}}, \quad \text{for } i \geq 0. \quad (8)$$

Write $S = \sum_{i \geq 0} \ell_i x^i$ and define the linear form L on $\mathbb{K}[x]$ by $L(x^i) = \ell_i$. Then Equation (6) is a direct consequence of (8).

For $i \geq 0$, equating coefficients of x^i, \dots, x^{2i-1} and x^{2i} in Equation (8) multiplied by F_i^* implies $L(F_i x^j) = 0$ for $i < j$ and $L(F_i x^i) = (-1)^i \gamma_{i+1} / \delta_{i+1}$. By linearity, this shows that L also satisfies Equality (5). \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.

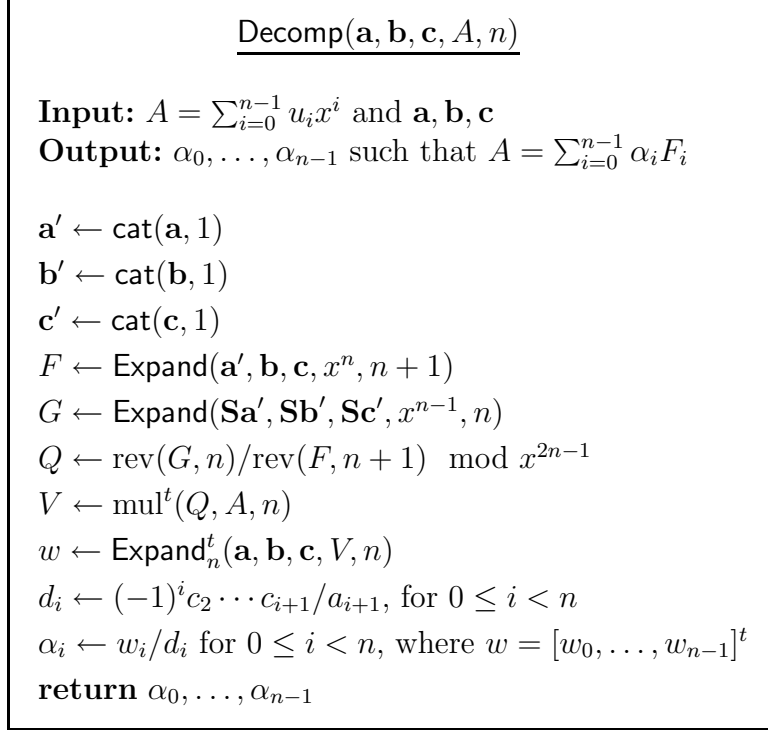


Fig. 3. Algorithm solving Problem Decomp_n

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 the previous section to compute $G = G_{n-1}$ and $F = F_n$ and we determine the power series expansion $\text{rev}(G_{n-1}, n) / \text{rev}(F_n, n+1) \pmod{x^{2n-1}}$. The first step takes $O(\mathbf{M}(n) \log(n))$ operations, and the second one $O(\mathbf{M}(n))$ using Newton iteration [10, Chap. 9]. In the pseudo-code we use the notation $\mathbf{S}\mathbf{x}$ 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 defined in the introduction, and let $\mathbf{H}_n = [H_{i,j}]_{0 \leq i, j < n}$ 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 , with $D_i = L(F_i^2)$. 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 (5) 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 $\mathbf{M}(n) + O(n)$. Using the transposed expansion algorithm `Expandt` of the previous section, multiplication by \mathbf{F}_n^t costs $O(\mathbf{M}(n) \log(n))$. Finally, multiplying by \mathbf{D}_n^{-1} takes linear time. This concludes the proof of Theorem 1.

Acknowledgments. This work was supported in part by the French National Agency for Research (ANR Project “Gecko”) and the Microsoft Research-INRIA Joint Centre.

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*. ACM, 2008. To appear.
- [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] J. R. Driscoll, Jr. D. M. Healy, 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.
- [7] Jean Favard. Sur les polynômes de Tchebicheff. *Comptes-Rendus de l’Académie des Sciences*, 200:2052–2053, 1935.
- [8] P. Flajolet. Combinatorial aspects of continued fractions. *Discrete Math.*, 32(2):125–161, 1980.
- [9] M. Frumkin. A fast algorithm for expansion over spherical harmonics. *Appl. Algebra Engrg. Comm. Comput.*, 6(6):333–343, 1995.
- [10] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1999.
- [11] J. Gerhard. Modular algorithms for polynomial basis conversion and greatest factorial factorization. In *RWCA’00*, pages 125–141, 2000.
- [12] G. Hanrot, M. Quercia, and P. Zimmermann. The Middle Product Algorithm, I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.
- [13] 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.
- [14] 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.

- [15] G. Leibon, D. Rockmore, and G. Chirikjian. A fast Hermite transform with applications to protein structure determination. In *SNC '07: Proceedings of the 2007 international workshop on Symbolic-numeric computation*, pages 117–124, New York, NY, USA, 2007. ACM.
- [16] 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.
- [17] D. Potts, G. Steidl, and M. Tasche. Fast algorithms for discrete polynomial transforms. *Math. Comp.*, 67(224):1577–1590, 1998.
- [18] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.
- [19] J. Shohat. The Relation of the Classical Orthogonal Polynomials to the Polynomials of Appell. *Amer. J. Math.*, 58(3):453–464, 1936.
- [20] 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.